Draft Standard for Information Technology—Standardized Application Environment Profile—POSIX Realtime and Embedded Application Support (AEP)

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

Unapproved draft

Abstract: This standard is part of the POSIX series of standardized profiles for open systems. It defines environment profiles for portable realtime and embedded applications.

Keywords: AEP, application portability, data processing environment, open systems, operating system, portable application, POSIX profiles, realtime application environments, realtime, embedded

IEEE P1003.13/D2.1
February 2003

IEEE Standards Activities Department
Standards Licensing and Contracts
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

3/2/03

Editor's Notes

This section will not appear in the final document. It is used for editorial comments concerning this draft. Please consult the balloting instructions document and the cover letter for the ballot that accompanied this draft for information on how the balloting process is accomplished.

This is the first draft of the POSIX.13 revision; POSIX.13-1998 defined four real-time application environment profiles (or POSIX subsets), based on the ISO/IEC 9945-1:1996 (POSIX.1) and the IEEE Std 1003.5c-1998 (POSIX.5c) standards. The goal of this revision is to update the profiles according to implementation experience, and to add the services defined in the new revised IEEE Std 1003.1-2001 (which incorporates among other services the recently approved POSIX amendments POSIX.1d, POSIX.1g, POSIX.1j, and POSIX.1q) and the POSIX.5c amendment. Also in the scope is to incorporate any new POSIX Ada bindings that might get developed and approved before the completion of this revision. The POSIX.13 revision project incorporates and supersedes work developed previously in the POSIX.13a and POSIX.13b projects

Changes to the previous standard have been marked with side bars like that affecting this sentence. These side bars are for information only. Small numbers are printed at the left margin of each page of the document to ease making references to specific text during the ballot process. These numbers may not match actual lines, and are only used as an approximate reference.

Please report typographical errors to:

Michael González Harbour
Dpto. de Electrónica y Computadores
Universidad de Cantabria
Avenida de los Castros s/n
39005 - Santander SPAIN
TEL: +34 942 201483
FAX: +34 942 201402
Email: `mgh@unican.es` (*Electronic mail is preferred.*)

The copying and distribution of IEEE balloting drafts is accomplished by the Standards Office. To report problems with reproduction of your copy, or to request additional copies of this draft, contact:

Tracy Woods
IEEE Computer Society,
1730 Massachusetts Avenue, NW,
Washington DC 20036-1992, USA.
Phone: +1-202-371-1013
Fax: +1-202-728-0884
E-mail: `twoods@computer.org`
Web page: `http://www.computer.org/standard/draftstd.htm`

POSIX.13 Change History

This section is provided to track major changes between drafts.

Draft 2 [July 2002] First ballot draft

Draft 1.2 [July 2002] First complete draft, for internal SSWG-RT use.

- Added new limits
- Eliminated requirement for reader/writer locks
- Added Annex B
- Added the alphabetical topical index
- Added requirement for priority ranges

Draft 1.1 [April 2002] Second draft for internal SSWG-RT use.

- Minor fixes and additions
- Incorporated some changes from discussions at the Open Group's Real-Time Forum.

Draft 1.0 [February 2002] First draft, incomplete, for internal SSWG-RT use.

- Scope of P1003.1a and P1003.1b (amendments to IEEE Std 1003.13-1998) included in this revision
- Targets the newly approved POSIX.1 revision (IEEE Std 1003.1-2001) and POSIX.5c
- Uses text developed for POSIX.13b, but reformatted as a revision to POSIX.13
- Standard's title changed to add support for embedded applications

Contents

List of Figures

List of Tables

Introduction

(This introduction is not a normative part of IEEE Std P1003.13, Information technology—Standardized Application Environment Profile—POSIX Realtime and Embedded Application Support (AEP).

The purpose of this standard is to define realtime and embedded application environments based on the ISO/IEC 9945 series of standards. It is intended for realtime systems implementors and realtime applications software developers.

This standard is a revision of IEEE Std 1003.13-1998, where four realtime application environment profiles (or POSIX subsets) are defined. The goal of this revision is to update each of the four profiles according to implementation experience, and to add the services defined in the newly approved POSIX standards:

— IEEE Std 1003.1-2001, Standard for *Information Technology—Portable Operating System Interface (POSIX)* (which includes among others the revised ISO/IEC 9945-1: 1996 as amended by IEEE Std 1003.1d-1999, IEEE Std 1003.1j-2000, IEEE Std 1003.1g-2000, and IEEE Std 1003.1q-2000)

— and the amendment to IEEE Std 1003.5-1992, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Programming Interface (API)* and IEEE Std 1003.5b-1996, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Binding for System Application Program Interface (API)—Amendment 1: Realtime Extensions*; this amendment is: IEEE Std 1003.5c-1998.

The base standard, IEEE Std 1003.1-2001, allows profiling standards supporting functional requirements less than those required in the full base standard to subset both mandatory and optional functionality required for POSIX Conformance (see the Base Definitions volume, Section 2.1.5.1, “Subprofiling Considerations”). The POSIX.13 standard articulates these subprofiling options through units of functionality, defined herein, and by use of named options defined in the base standard.

This standard specifies four realtime profiles both for the C Language and for the Ada Language options. Because Ada Bindings to IEEE Std 1003.1-2001 are currently under development, the C Language option contains more services than the Ada Language option in the current draft. If these Ada Bindings are completed before this proposed standard is sent to ballot, the draft will be amended to incorporate them. Otherwise, an amendment of IEEE 1003.13 will be produced in the future, to incorporate the added Ada Language services.

This standard is designed to support building systems where not all the interconnected boxes use the same profile, for example, a hierarchical system where the

bottom-level device controllers use the “minimal” profile, the next level up follows the larger “control” profile, and so on. There are interfaces called out for the smaller profiles that make no sense in an isolated box; those interfaces are there solely to support the construction of heterogeneous systems, and systems of communicating peers. Such systems are very common in practice.

To summarize, this standard is embedded in a much larger and widely supported set of standards, which yields benefits during code development, as much development and testing is done on the larger and more comfortable systems. It also may be used in the construction of large and heterogeneous systems.

Four profiles have been defined to reflect the wide range of system requirements presented by realtime designs. The intent is to provide a meaningful and coherent set of interfaces that will provide software vendors and consumers with a uniform framework for describing and specifying operating system capabilities. This allows an application writer to construct an application that may be easily moved to a different system that supports the same profile. Similarly, it allows a vendor to claim conformance with an established standard, even if that vendor's implementation does not support the full POSIX feature set.

Initially, the focus of this standard is to provide standardized environments supporting the C language. Options are provided for bindings to the Ada programming language as well as for the C language. Bindings for other languages to these services may be developed and this standard will be updated as appropriate.

Within this document, the term “POSIX.13” refers to this standard, IEEE Std 1003.13-200x.

Editor’s note: 200x will be changed to match the year the 1003.13 revision is approved as a standard.

Organization of This Standard

This Standard is divided into eight elements:

(1) General (Section 1)

(2) Normative references (Section 2)

(3) Definitions (Section 3)

(4) Conventions and abbreviations (Section 4)

(5) Conformance (Section 5)

(6) The various realtime profiles (Sections 6 through 9)

(7) ISPICS requirements (C) (Annex A)

(8) ISPICS requirements (Ada) (Annex B)

References are provided to direct the reader to other related sections.

Informative annexes are not normative parts of the standard and are provided for information only. They are provided for guidance and to help understanding.

In publishing this Standard, its developers simply intend to provide a yardstick against which various operating system implementations can be measured for conformance. It is not the intent of the developers to measure or rate any products, to reward or sanction any vendors of products for conformance or lack of conformance to this Standard, or to attempt to enforce this Standard by these or any other means. The responsibility for determining the degree of conformance or lack thereof with this Standard rests solely with the individual who is evaluating a product claiming to be in conformance with this Standard.

Base Documents

The various realtime application environments described herein are based on the ISO/IEC 9945 and IEEE 1003 family of documents as well as ISO 9899 (C99 Language) and 8652 (Ada95 Language).

Scenario

This standard is based directly on existing small and/or realtime (typically non-UNIX™[1]) kernel practice as well as the growing body of practice with POSIX conformant kernels having realtime features. The general approach taken in this standard is to specify interfaces (taken from POSIX) sufficient to deliver the functionality typical of current realtime systems, (see Table 1-19 through Table 1-21).

Each profile is specified with full features, to give users clear direction. Vendors may provide means to configure out those parts that are not needed by specific applications. Vendors wishing to expand on the specified profiles are strongly encouraged to take the added interfaces from other POSIX.13 profiles or from the base standards, rather than invent new interfaces.

For each profile, the minimum hardware typically required is specified. This is the hardware assumed to be present; implementations may, of course, have more, but nothing in the profile requires—either directly or indirectly—more than the specified minimum hardware model.

Audience

The intended audience for this class of profiles is all persons concerned with an industry-wide standard realtime application environment based on the POSIX suite of standards. This includes at least four groups of people:

1. UNIX is a registered trademark of The Open Group in the United States of America and other countries.

(1) Persons buying hardware and software systems.

(2) Persons managing companies that are deciding on future corporate computing directions.

(3) Persons implementing realtime operating systems.

(4) Persons developing realtime applications where portability is a primary objective.

Rationale on Background

This subclause contains rationale common to all four realtime profiles.

The developers of POSIX.13 represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others. In the course of their deliberations, the developers reviewed related U.S. and international standards, both published and in progress.

Conceptually, POSIX.13 describes a set of application environment profiles needed for the construction and execution of portable realtime application programs.

The developers of this standard have tried to capture the functionality of existing realtime systems in a reasonable number of profiles that specify predominate application environments. It is felt that these profiles, although not optimum, are a best fit to existing classes of applications and systems.

Features of several commercial realtime kernels were considered during the development of the 1998 version of POSIX.13. These included **pSOS**™[1], **VRTX32**™[2], and **VxWorks**™[3]. Since these products were commercially successful, they must have addressed a significant market segment. In addition, the uniprocessor subset of VITA's **ORKID** specification, NGCR's "**Tiny Real Time**" (**TRT**), and the **uITRON** specification were examined. These were all proposed standard interfaces for small realtime embedded systems.

Features of other commercial realtime kernels such as **RT-Linux**[4] and **QNX**[5], as well as free software products such as **RTEMS**[6] were considered during the development of the current revision of POSIX.13.

The following is a list of features that are representative of current realtime systems and highlights the range of system requirements. While some concepts are

1. **pSOS** is now a registered trademark of Wind River Systems, Inc.
2. **VRTX32** is now a registered trademark of Mentor Graphics.
3. **VxWorks** is a registered trademark of Wind River Systems, Inc.
4. **RT-Linux** is... FSM Labs
5. **QNX** is ...
6. **RTEMS** is ...

common to virtually all implementations (e.g., preemptive, priority-based scheduling), some only apply to smaller systems (e.g., a single address space), and some only to more full-featured systems (e.g., network support, self-hosting).

Basic Realtime Multitasking and Synchronization

- Multiple flows of control
- Preemptive priority scheduling of flows of control
- One address space for all flows of control
- Direct control of location of memory areas
- Inter-thread communications mechanism via message passing (queues)
- Binary and counting semaphores, without priority inheritance
- Mutual exclusion, with optional priority inheritance or priority ceiling protocols
- Local or global event flags (one thread awaits multiple things)
- Multiple memory areas, with both fixed- and variable-sized block allocation policies
- System time in units of clock ticks
- Timeouts on all blocking services in units of clock ticks
- Hardware interrupt control and support for user interrupt handlers
- Signals
- Exception handling
- Minimal synchronous I/O interface: *open*(), *close*(), *read*(), *write*(), *ioctl*()
- Debugger interface
- No memory protection
- Application runs in privileged (supervisor) mode, if applicable
- Direct I/O, rather than via kernel
- System executable size and memory requirements are major constraints

I/O

Realtime systems supporting I/O generally provide the following features:

- Named I/O devices

— Support for serial I/O lines

— Pipes

— Installable user device drivers

— Memory mapped I/O

Local File System

Realtime systems supporting a file system generally provide the following features:

— Named files

— Hierarchical filesystem (directories)

— Contiguous preallocation of disk space

— May provide media compatibility with another filesystem (e.g. **MSDOS**™[1], or **RT-11**™[2])

— No user IDs or file protection

Historically, filesystems for embedded realtime systems typically have had a one-level name space, contiguous allocation of disk space, and relatively short filenames. They have not supported an arbitrary hierarchy of named directories, non-contiguous allocation of disk space, or long filenames. They may have had numbered directories (e.g. **RSX-11M**™[3]), or only contiguous allocation of disk space (e.g., **RT-11**™)

However, recent commercial offerings have supported multilevel named directories and both contiguous and non-contiguous disk space allocation. In these implementations, the support of these features with potentially non-deterministic performance does not preclude an application from restricting itself to features with deterministic performance. For example, it is still possible to use contiguous files exclusively. Because it is relatively easy to implement both, and need not interfere with deterministic performance, the working group did not make a distinction between realtime and time-sharing file systems in this AEP.

Although few embedded systems had a hard drive and a file system, present flash memory technology has enabled embedded systems, even those with strict vibration requirements, to have a file system resident on this kind of non-volatile media. This has caused the POSIX.13 profile designed for large embedded systems, the Dedicated Realtime System Profile (PSE53), to incorporate a simplified file system in this new revision of the standard.

1. **MS-DOS** is a registered trademark of Microsoft Corporation.
2. **RT-11** is now a registered trademark of Compaq.
3. **RSX-11M** is now a registered trademark of Compaq.

Traditional implementations of POSIX.1 filesystems employ a disk buffer cache to improve average performance by reducing the number of physical media accesses, and by reordering the accesses to take advantage of the characteristics of rotating media. These implementations have not made a distinction between the buffering of data transfers [*read*() and *write*()], and directory operations [*creat*(), *link*(), *unlink*(), *mkdir*(), *rmdir*(), *rename*()]. A result of this is that a system crash at an unexpected moment can leave the filesystem in a corrupted state. This situation is usually corrected at the next system reboot by a filesystem checker and recovery program, such as `fsck`. The checking and correcting of a corrupted filesystem may take a long and variable amount of time to perform, may require a human operator to monitor and control its progress, and may nonetheless fail to repair the filesystem. Any one of these characteristics would make a filesystem check unacceptable for some embedded realtime applications. It was therefore suggested that such applications limit their use of directory operations to *safe* times, and that implementations maintain the filesystem in such a way that a filesystem check during reboot is avoided. This was considered, but rejected on the grounds that not all applications would require the capability, and that it was neither specifiable nor testable.

Network Communication

Realtime systems supporting networking generally provide the following features:

— Compatibility with a protocol stack (e.g. TCP/IP)

— May support applications such as FTP, TELNET, TFTP, rcp

Distributed File System

Realtime systems supporting a distributed (non-local) file system generally provide the following features:

— Remote access to a filesystem

— Performance not realtime

Memory Protection

Realtime systems supporting memory protection (typically requiring a memory management unit) generally provide the following features:

— Memory mapping and protection

— Ability to map to special areas of memory (I/O page, frame buffer)

— Typically do not have demand paging for realtime parts

Multiprocessor Support

Realtime systems supporting multiprocessing generally provide one of the following methods:

— **network**
Non-transparent access to remote objects, remote procedure calls

— **distributed**
Transparent access to objects, no load-balancing

— **symmetric**
Presence of a global task scheduling queue (may also have local scheduling queues)

Self-Hosting

Realtime systems supporting the capability for program development, text editing, compilation, etc. generally provide the following features:

— Shell

— Text editor

— Compiler, assembler, linker, debugger

— May have user ID protection

Only the larger profiles (i.e., PSE54) are likely to be self-hosted.

Overview of the Profiles Structure (Rationale)

This subclause contains rationale common to all four realtime profiles.

The four profiles defined in this standard are designed to make applications upwards compatible to higher profiles. Figure I.1 shows the main building blocks of each of the four profiles specified in this standard. Please note that the full differences between the different profiles are more complex than those appearing on this figure. See subclause 1.6, “Summary of Profile Features”, for a full description of the differences between the profiles.

The “core” building block in Figure I.1 refers to the units of functionality and options required in all four profiles. See subclause 6.2, “Operating System Interface Requirements”, for a description of the core services. Profiles with only one implicit process (PSE51 and PSE52) are shaded in the figure, to highlight this major difference with the larger profiles, which require support for multiple processes (and thus require having a MMU).

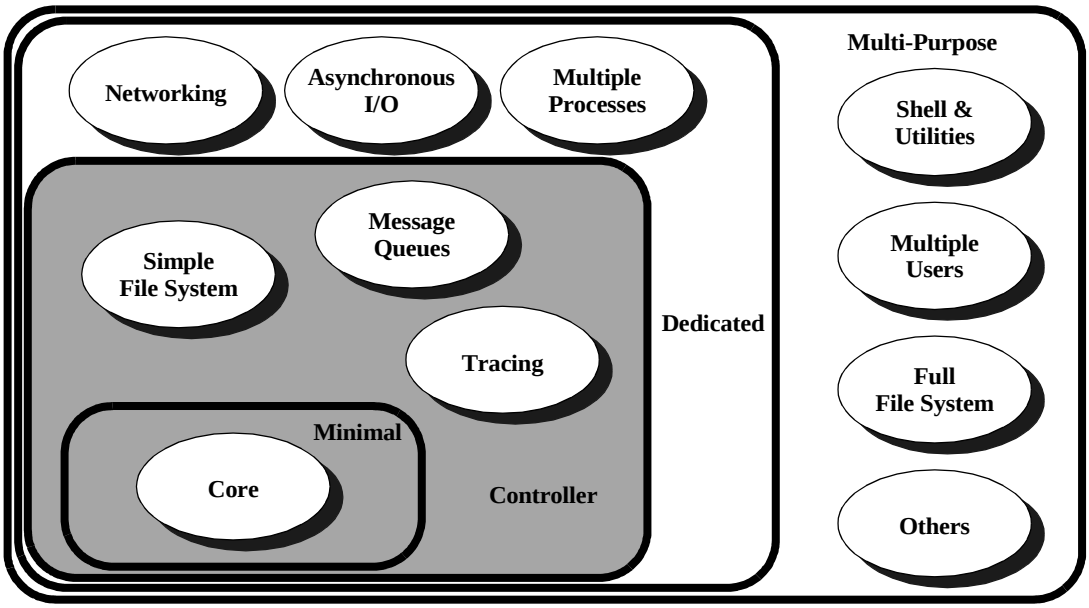

Figure I.1. Main Building Blocks of the Profiles

Related Standards Activities

Activities to extend this Standard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time or are expected to become active in the near future[1]:

(1) Additional system application program interfaces (APIs) in C language

(2) Ada language bindings

(3) Additional realtime facilities

(4) Fault tolerance

(5) Profiles describing application- or user-specific combinations of Open Systems standards

1. A Standards Status Report that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone: +1 202 371-0101; FAX: +1 202 728-9614. Working drafts of POSIX standards under development are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331 (http://www.standards.ieee.org/).

If you have interest in participating in the Portable Application Standards Committee (PASC) working groups addressing these issues, please send your name, address, and phone number to

Secretary, IEEE Standards Board
Institute of Electrical and Electronics Engineers, Inc.
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331
USA

When writing, ask to have your letter forwarded to the chairperson of the appropriate PASC working group.

If you have interest in participating in this work at the international level, contact your International Organization for Standardization/International Electrotechnical Committee (ISO/IEC) national body.

IEEE Std 1003.13-1998 was prepared by the System Services Working Group—Realtime, sponsored by the Portable Application Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the System Services Working Group—Realtime was as follows:

Portable Application Standards Committee

Chair:	Lowell Johnson
Vice Chair:	Joseph M. Gwinn
Functional Vice Chairs:	Jay Ashford Andrew Josey Curtis Royster, Jr.
Secretary:	Nick Stoughton

IEEE System Services Working Group—Realtime

Chair:	Joseph M. Gwinn Susan Corwin (to 1995)
Secretary:	Karen D. Gordon Frank Prindle (1996) Lee Schermerhorn (to 1994)
Technical Editor:	Bob Luken
Ballot Coordinators:	Andrew E. Wheeler, Jr. James T. Oblinger
Technical Reviewers:	Andrew E. Wheeler, Jr. Joseph M. Gwinn Karen D. Gordon

Working Group

Ray Alderman	Michael Feustel	Dave Lunger
Larry Anderson	Bill Gallmeister	Bill Maes
Pierre-Jean Arcos	Michael González	James T. Oblinger
Charles R. Arnold	Karen D. Gordon	Offer Pazy
V. Raj Avula	Randy Greene	Carolyn Petersen
Theodore P. Baker	Rick Greer	Dave Plauger
Todd Bargorek	Joseph M. Gwinn	Arlan Pool
Robert Barned	Steven A. Haaser	Franklin C. Prindle
Richard M. Bergman	Barbara Haleen	François Riche
Nawaf Bitar	Geoffrey R. Hall	Robert Rose

Steve Brosky	Patrick Hebert	Gordon W. Ross
David Butenhof	Mary R. Hermann	Barry Ruzek
Hans Petter Christiansen	David Hughes	Webb Scales
Dave Cooper	Duane Hughes	Lee Schermerhorn
Susan Corwin	Michael B. Jones	Del Swanson
Bill Cox	Steven Kleiman	Barry Traylor
June R. Curtis	Robert Knighten	Stephen R. Walli
Peter Dibble	C. Douglass Locke	Andrew E. Wheeler, Jr.
Christoph Eck	Kent Long	David Wilner
Maryland R. Edwards	Robert D. Luken	John Zolnowsky

Ballot Group

The following persons were members of the 1003.13 Balloting Group that approved the standard for submission to the IEEE Standards Board:

Norman Aaronson	John Gilbert	Robert D. Luken
Alejandro Alonso-Muñoz	Michael González	Dave Lunger
Pierre-Jean Arcos	Karen D. Gordon	Marshall McKusick
Charles R. Arnold	Mars J. Gralia	Craig B. Meyers
Theodore P. Baker	Randy Greene	Diana Norwood
Robert Barned	Joseph M. Gwinn	James T. Oblinger
Jason Behm	Steven A. Haaser	Dave Plauger
Richard M. Bergman	Geoffrey R. Hall	Arlan Pool
Andy R. Bihain	Patrick Hebert	Franklin C. Prindle
Shirley Bockstahler-Brandt	Hans H. Heilborn	Paul Rabin
Steve Case	Duane Hughes	Wendy Rauch
Hu Cheng	Hal Jespersen	Henry H. Robbins
Hans Pietter Christiansen	Michael B. Jones	Steven Schwarm
Susan Corwin	Joe Kelsey	Del Swanson
Donald Cragun	Judy Kerner	Sandra Swearingen
June R. Curtis	Lawrence J. Kilgallen	James G. Tanner
Lee Damico	Martin J. Kirk	Mark-Rene Uchida
Christoph Eck	Thomas M. Kurihara	Andrew E. Wheeler, Jr.
James A. Eiler	Kevin Lewis	David Wilner
Philip H. Enslow	C. Douglass Locke	Oren Yuen
Donna K. Fisher	Kent Long	John J. Zenor
Michel Gien	James P. Lonjers	John Zolnowsky
	Lee W. Lucas	

When the IEEE-SA Standards Board approved IEEE 1003.13 on 19 March 1998, it had the following membership:

Richard J. Holleman, *Chair*	**Judith Gorman**, *Secretary*	**Donald N. Heirman**, *Vice Chair*

Satish K. Aggarwal
Clyde R. Camp
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster[a]
Thomas F. Garrity
Ruben D. Garzon
James H. Gurney
Jim D. Isaak
Lowell G. Johnson
Robert Kennelly
E.G. "Al" Kiener
Joseph L. Koepfinger[a]
Stephen R. Lambert
Jim Logothetis
Donald C. Loughry
L. Bruce McClung
Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Hans E. Weinrich
Donald W. Zispe

a. Member emeritus

Noelle Humenick
IEEE Standards Project Editor

IEEE P1003.13 was prepared by the System Services Working Group—Realtime, sponsored by the Portable Application Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the System Services Working Group—Realtime was as follows:

Portable Application Standards Committee

Chair:	Lowell Johnson
Vice Chair:	Joseph M. Gwinn
Functional Vice Chairs:	Jay Ashford Andrew Josey Curtis Royster Jr.
Secretary:	Nick Stoughton

IEEE System Services Working Group—Realtime

Chair:	Joseph M. Gwinn
Secretary:	Karen D. Gordon
Technical Editor:	Michael González
Ballot Coordinator:	Jim Oblinger
Technical Reviewers:	Michael González

Working Group

<to be added later>

Ballot Group

<to be added later>

Draft Standard for Information Technology—Standardized Application Environment Profile —POSIX Realtime and Embedded Application Support (AEP)

Section 1: Overview

1.1 Scope

This standard establishes a set of Realtime and Embedded Environment Profiles based on IEEE Std 1003.1-2001, IEEE Std 1003.5-1992 as amended by IEEE Std 1003.5b-1996 and IEEE Std 1003.5c-1998, and related standards specifying foundations for realtime applications. It is a revision of the previous IEEE Std 1003.13-1998, which established Realtime Profiles based on ISO/IEC 9945-1:1990 as amended by IEEE Std 1003.1b-1993, IEEE Std 1003.5b, and ISO/IEC 9945-2:1993.

The Application Environment Profiles specified herein are appropriate for the development and execution of realtime or embedded applications using the services and utilities provided by standards called out in this document.

1.2 Taxonomy Position

P— OSE Profiles

AEP— Application Environment Profiles

PS— System Profiles

PSE— Generic Environment Profiles

PSE5— Realtime Environments

PSE51— Minimal Realtime System Profile

PSE52— Realtime Controller System Profile

PSE53— Dedicated Realtime System Profile

PSE54— Multi-Purpose Realtime System Profile

1.2.1 Rationale for Positioning (informative)

(This subclause is not a normative part of IEEE Std P1003.13)

This document contains requirements for Application Program Interfaces and Units of Functionality necessary to support four instances of the Generic Realtime Environment class of applications. It specifies the behavior to be observed at the interfaces of the Application Platform on which the class of applications can run. This subset of an OSE profile is complete and coherent within the context of the class of applications supported. As such, it is a System Profile class of Application Environment Profile (AEP).

1.3 Realtime System Profiles

This document describes four realtime profiles and their minimum hardware requirements.

1.3.1 Minimal Realtime System Profile (PSE51)

These systems are typically embedded in systems dedicated to unattended control of one or more special I/O devices. Neither user interaction nor a file system (mass storage) is required. The programming model is that of a single (implicit) POSIX process (corresponding to the processor's hardware address space) containing one or more threads of control (POSIX.1 threads or Ada tasks). Although there is only one process, a Message Passing interface is provided for communications among threads of control and between PSE5X instantiations. Special devices are operated

and controlled either by memory-mapped I/O or by the basic I/O interface, which provides a standard way to access the intrinsically nonstandard I/O hardware and its non-portable control code.

The hardware model for this profile assumes a single processor with its memory, but no memory management unit (MMU) or common I/O devices are required. (If there are in fact multiple processors, typically there are multiple instantiations of the operating system, perhaps communicating via shared memory or a backplane channel, perhaps isolated).

1.3.2 Realtime Controller System Profile (PSE52)

These systems are an extension of the Minimal Realtime System Profile. Support for a file system interface and asynchronous (non-blocking) I/O interfaces has been added.

The hardware model for this profile assumes a single processor and memory space (a MMU is not required). Mass storage devices are not required; the file system may, for instance, be implemented in memory (RAM disk or flash memory).

1.3.3 Dedicated Realtime System Profile (PSE53)

These systems are an extension of the Realtime Controller System Profile. Support for multiple processes has been added. Although these are usually embedded systems, flash memory technology enables presence of a simplified file system, even in those systems with mechanical or environmental requirements that preclude a rotating-media hard drive. Since memory management hardware may be provided, the functionality of memory locking is provided.

The hardware model for this profile assumes one or more processors, each with its own MMU, in the same system.

1.3.4 Multi-Purpose Realtime System Profile (PSE54)

These systems include all the functionality of the other three profiles. They provide comprehensive functionality and run a mix of differing realtime and non-realtime tasks. This functionality includes most of POSIX.1 and/or POSIX.5c. Since users may conduct interactive sessions on those systems, all the mandatory elements of the Shell and Utilities volume of POSIX.1 are also included. Support for multiple multi-threaded processes is required so that multi-tasking may be done by threads (POSIX.1 threads or Ada tasks), processes, or both.

The hardware model for this profile assumes one or more processors with memory management units, high-speed storage devices, special interfaces, network support, and display devices. The system supports a mix of realtime and non-realtime tasks, some being interactive user tasks.

1.4 Units of Functionality

Some of the profiles specified in this standard do not require support for all the functionality specified in a referenced standard. In this case, if that referenced standard does not contain options for specifying just the required functionality, only those Units of Functionality referenced by the profile may be used by a strictly conforming application.

Table 1-1 shows the Units of Functionality defined for POSIX.1; each of these units represents a Subprofiling Option Group (See the Base Definitions Volume of POSIX.1 {3}, Section 2.1.5.1, “Subprofiling Considerations”), and is a set of functions that represents a separately implementable element of POSIX.1. Table 1-2 through Table 1-18 show the Units of Functionality defined for POSIX.5c.

Table 1-1: POSIX.1 Units of Functionality

Unit of Functionality	Included Functions
POSIX_C_LANG_JUMP	*longjmp*(), *setjmp*()

Table 1-1: POSIX.1 Units of Functionality (Continued)

Unit of Functionality	Included Functions
POSIX_C_LANG_MATH	*acos*(), *acosf*(), *acosh*(), *acoshf*(), *acoshl*(), *acosl*(), *asin*(), *asinf*(), *asinh*(), *asinhf*(), *asinhl*(), *asinl*(), *atan*(), *atan2*(), *atan2f*(), *atan2l*(), *atanf*(), *atanh*(), *atanhf*(), *atanhl*(), *atanl*(), *cabs*(), *cabsf*(), *cabsl*(), *cacos*(), *cacosf*(), *cacosh*(), *cacoshf*(), *cacoshl*(), *cacosl*(), *carg*(), *cargf*(), *cargl*(), *casin*(), *casinf*(), *casinh*(), *casinhf*(), *casinhl*(), *casinl*(), *catan*(), *catanf*(), *catanh*(), *catanhf*(), *catanhl*(), *catanl*(), *cbrt*(), *cbrtf*(), *cbrtl*(), *ccos*(), *ccosf*(), *ccosh*(), *ccoshf*(), *ccoshl*(), *ccosl*(), *ceil*(), *ceilf*(), *ceill*(), *cexp*(), *cexpf*(), *cexpl*(), *cimag*(), *cimagf*(), *cimagl*(), *clog*(), *clogf*(), *clogl*(), *conj*(), *conjf*(), *conjl*(), *copysign*(), *copysignf*(), *copysignl*(), *cos*(), *cosf*(), *cosh*(), *coshf*(), *coshl*(), *cosl*(), *cpow*(), *cpowf*(), *cpowl*(), *cproj*(), *cprojf*(), *cprojl*(), *creal*(), *crealf*(), *creall*(), *csin*(), *csinf*(), *csinh*(), *csinhf*(), *csinhl*(), *csinl*(), *csqrt*(), *csqrtf*(), *csqrtl*(), *ctan*(), *ctanf*(), *ctanh*(), *ctanhf*(), *ctanhl*(), *ctanl*(), *erf*(), *erfc*(), *erfcf*(), *erfcl*(), *erff*(), *erfl*(), *exp*(), *exp2*(), *exp2f*(), *exp2l*(), *expf*(), *expl*(), *expm1*(), *expm1f*(), *expm1l*(), *fabs*(), *fabsf*(), *fabsl*(), *fdim*(), *fdimf*(), *fdiml*(), *floor*(), *floorf*(), *floorl*(), *fma*(), *fmaf*(), *fmal*(), *fmax*(), *fmaxf*(), *fmaxl*(), *fmin*(), *fminf*(), *fminl*(), *fmod*(), *fmodf*(), *fmodl*(), *fpclassify*(), *frexp*(), *frexpf*(), *frexpl*(), *hypot*(), *hypotf*(), *hypotl*(), *ilogb*(), *ilogbf*(), *ilogbl*(), *isfinite*(), *isgreater*(), *isgreaterequal*(), *isinf*(), *isless*(), *islessequal*(), *islessgreater*(), *isnan*(), *isnormal*(), *isunordered*(), *ldexp*(), *ldexpf*(), *ldexpl*(), *lgamma*(), *lgammaf*(), *lgammal*(), *llrint*(), *llrintf*(), *llrintl*(), *llround*(), *llroundf*(), *llroundl*(), *log*(), *log10*(), *log10f*(), *log10l*(), *log1p*(), *log1pf*(), *log1pl*(), *log2*(), *log2f*(), *log2l*(), *logb*(), *logbf*(), *logbl*(), *logf*(), *logl*(), *lrint*(), *lrintf*(), *lrintl*(), *lround*(), *lroundf*(), *lroundl*(), *modf*(), *modff*(), *modfl*(), *nan*(), *nanf*(), *nanl*(), *nearbyint*(), *nearbyintf*(), *nearbyintl*(), *nextafter*(), *nextafterf*(), *nextafterl*(), *nexttoward*(), *nexttowardf*(), *nexttowardl*(), *pow*(), *powf*(), *powl*(), *remainder*(), *remainderf*(), *remainderl*(), *remquo*(), *remquof*(), *remquol*(), *rint*(), *rintf*(), *rintl*(), *round*(), *roundf*(), *roundl*(), *scalbln*(), *scalblnf*(), *scalblnl*(), *scalbn*(), *scalbnf*(), *scalbnl*(), *signbit*(), *sin*(), *sinf*(), *sinh*(), *sinhf*(), *sinhl*(), *sinl*(), *sqrt*(), *sqrtf*(), *sqrtl*(), *tan*(), *tanf*(), *tanh*(), *tanhf*(), *tanhl*(),*tanl*(), *tgamma*(), *tgammaf*(),*tgammal*(), *trunc*(), *truncf*(), *truncl*()

Table 1-1: POSIX.1 Units of Functionality (Continued)

Unit of Functionality	Included Functions
POSIX_C_LANG_SUPPORT	*abs*(), *asctime*(), *asctime_r*(), *atof*(), *atoi*(), *atol*(), *atoll*(), *bsearch*(), *calloc*(), *ctime*(), *ctime_r*(), *difftime*(), *div*(), *feclearexcept*(), *fegetenv*(), *fegetexceptflag*(), *fegetround*(), *feholdexcept*(), *feraiseexcept*(), *fesetenv*(), *fesetexceptflag*(), *fesetround*(), *fetestexcept*(), *feupdateenv*(), *free*(), *gmtime*(), *gmtime_r*(), *imaxabs*(), *imaxdiv*(), *isalnum*(), *isalpha*(), *isblank*(), *iscntrl*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*(), *labs*(), *ldiv*(), *llabs*(), *lldiv*(), *localeconv*(), *localtime*(), *localtime_r*(), *malloc*(), *memchr*(), *memcmp*(), *memcpy*(), *memmove*(), *memset*(), *mktime*(), *qsort*(), *rand*(), *rand_r*(), *realloc*(), *setlocale*(), *snprintf*(), *sprintf*(), *srand*(), *sscanf*(), *strcat*(), *strchr*(), *strcmp*(), *strcoll*(), *strcpy*(), *strcspn*(), *strerror*(), *strerror_r*(), *strftime*(), *strlen*(), *strncat*(), *strncmp*(), *strncpy*(), *strpbrk*(), *strrchr*(), *strspn*(), *strstr*(), *strtod*(), *strtof*(), *strtoimax*(), *strtok*(), *strtok_r*(), *strtol*(), *strtold*(), *strtoll*(), *strtoul*(), *strtoull*(), *strtoumax*(), *strxfrm*(), *time*(), *tolower*(), *toupper*(), *tzname*, *tzset*(), *va_arg*(), *va_copy*(), *va_end*(), *va_start*(), *vsnprintf*(), *vsprintf*(), *vsscanf*()
POSIX_C_LANG_WIDE_CHAR	*btowc*(), *iswalnum*(), *iswalpha*(), *iswblank*(), *iswcntrl*(), *iswctype*(), *iswdigit*(), *iswgraph*(), *iswlower*(), *iswprint*(), *iswpunct*(), *iswspace*(), *iswupper*(), *iswxdigit*(), *mblen*(), *mbrlen*(), *mbrtowc*(), *mbsinit*(), *mbsrtowcs*(), *mbstowcs*(), *mbtowc*(), *swprintf*(), *swscanf*(), *towctrans*(), *towlower*(), *towupper*(), *vswprintf*(), *vswscanf*(), *wcrtomb*(), *wcscat*(), *wcschr*(), *wcscmp*(), *wcscoll*(), *wcscpy*(), *wcscspn*(), *wcsftime*(), *wcslen*(), *wcsncat*(), *wcsncmp*(), *wcsncpy*(), *wcspbrk*(), *wcsrchr*(), *wcsrtombs*(), *wcsspn*(), *wcsstr*(), *wcstod*(), *wcstof*(), *wcstoimax*(), *wcstok*(), *wcstol*(), *wcstold*(), *wcstoll*(), *wcstombs*(), *wcstoul*(), *wcstoull*(), *wcstoumax*(), *wcsxfrm*(), *wctob*(), *wctomb*(), *wctrans*(), *wctype*(), *wmemchr*(), *wmemcmp*(), *wmemcpy*(), *wmemmove*(), *wmemset*()
POSIX_DEVICE_IO	*clearerr*(), *close*(), *fclose*(), *fdopen*(), *feof* (), *ferror*(), *fflush* (), *fgetc*(), *fgets*(), *fileno*(), *fopen*(), *fprintf*(), *fputc*(), *fputs*(), *fread*(), *freopen*(), *fscanf*(), *fwrite*(), *getc*(), *getchar*(), *gets*(), *open*(), *perror*(), *printf*(), *putc*(), *putchar*(), *puts*(), *read*(), *scanf*(), *setbuf*(), *setvbuf*(), *ungetc*(), *vfprintf* (), *vfscanf*(), *vprintf*(), *vscanf*(), *write*()

Table 1-1: POSIX.1 Units of Functionality (Continued)

Unit of Functionality	Included Functions
POSIX_DEVICE_SPECIFIC	*cfgetispeed*(), *cfgetospeed*(), *cfsetispeed*(), *cfsetospeed*(), *ctermid*(), *isatty*(), *tcdrain*(), *tcflow*(), *tcflush*(), *tcgetattr*(), *tcsendbreak*(), *tcsetattr*(), *ttyname*(), *ttyname_r*()
POSIX_EVENT_MGMT	*FD_CLR*(), *FD_ISSET*(), *FD_SET*(), *FD_ZERO*(), *pselect*(), *select*()
POSIX_FD_MGMT	*dup*(), *dup2*(), *fcntl*(), *fgetpos*(), *fseek*(), *fseeko*(), *fsetpos*(), *ftell*(), *ftello*(), *ftruncate*(), *lseek*(), *rewind*()
POSIX_FIFO	*mkfifo*()
POSIX_FILE_ATTRIBUTES	*chmod*(), *chown*(), *fchmod*(), *fchown*(), *umask*()
POSIX_FILE_LOCKING	*flockfile*(), *ftrylockfile*(), *funlockfile*(), *getc_unlocked*(), *getchar_unlocked*(), *putc_unlocked*(), *putchar_unlocked*()
POSIX_FILE_SYSTEM	*access*(), *chdir*(), *closedir*(), *creat*(), *fpathconf*(), *fstat*(), *getcwd*(), *link*(), *mkdir*(), *opendir*(), *pathconf*(), *readdir*(), *readdir_r*(), *remove*(), *rename*(), *rewinddir*(), *rmdir*(), *stat*(), *tmpfile*(), *tmpnam*(), *unlink*(), *utime*()
POSIX_FILE_SYSTEM_EXT	*glob*(), *globfree*()
POSIX_JOB_CONTROL[a]	*setpgid*(), *tcgetpgrp*(), *tcsetpgrp*()
POSIX_MULTI_PROCESS	*_Exit*(), *_exit*(), *assert*(), *atexit*(), *clock*(), *execl*(), *execle*(), *execlp*(), *execv*(), *execve*(), *execvp*(), *exit*(), *fork*(), *getpgrp*(), *getpid*(), *getppid*(), *setsid*(), *sleep*(), *times*(), *wait*(), *waitpid*()
POSIX_NETWORKING	*accept*(), *bind*(), *connect*(), *endhostent*(), *endnetent*(), *endprotoent*(), *endservent*(), *freeaddrinfo*(), *gai_strerror*(), *getaddrinfo*(), *gethostbyaddr*(), *gethostbyname*(), *gethostent*(), *gethostname*(), *getnameinfo*(), *getnetbyaddr*(), *getnetbyname*(), *getnetent*(), *getpeername*(), *getprotobyname*(), *getprotobynumber*(), *getprotoent*(), *getservbyname*(), *getservbyport*(), *getservent*(), *getsockname*(), *getsockopt*(), *htonl*(), *htons*(), *if_freenameindex*(), *if_indextoname*(), *if_nameindex*(), *if_nametoindex*(), *inet_addr*(), *inet_ntoa*(), *inet_ntop*(), *inet_pton*(), *listen*(), *ntohl*(), *ntohs*(), *recv*(), *recvfrom*(), *recvmsg*(), *send*(), *sendmsg*(), *sendto*(), *sethostent*(), *setnetent*(), *setprotoent*(), *setservent*(), *setsockopt*(), *shutdown*(), *socket*(), *sockatmark*(), *socketpair*()
POSIX_PIPE	*pipe*()
POSIX_PRIORITY_RANGES	*sched_get_priority_max*(), *sched_get_priority_min*(), *sched_rr_get_interval*()
POSIX_REGEXP[b]	*regcomp*(), *regerror*(), *regexec*(), *regfree*()

Table 1-1: POSIX.1 Units of Functionality (Continued)

Unit of Functionality	Included Functions
POSIX_RW_LOCKS[c]	*pthread_rwlock_destroy*(), *pthread_rwlock_init*(), *pthread_rwlock_rdlock*(), *pthread_rwlock_timedrdlock*()[d], *pthread_rwlock_timedwrlock*()[d], *pthread_rwlock_tryrdlock*(), *pthread_rwlock_trywrlock*(), *pthread_rwlock_unlock*(), *pthread_rwlock_wrlock*(), *pthread_rwlockattr_destroy*(), *pthread_rwlockattr_getpshared*()[e], *pthread_rwlockattr_init*(), *pthread_rwlockattr_setpshared*()[e]
POSIX_SHELL_FUNC	*pclose*(), *popen*(), *system*(), *wordexp*(), *wordfree*()
POSIX_SIGNALS	*abort*(), *alarm*(), *kill*(), *pause*(), *raise*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigemptyset*(), *sigfillset*(), *sigismember*(), *signal*(), *sigpending*(), *sigprocmask*(), *sigsuspend*(), *sigwait*()
POSIX_SIGNAL_JUMP	*siglongjmp*(), *sigsetjmp*()
POSIX_SINGLE_PROCESS	*confstr*(), *getenv*(), *setenv*(), *sysconf*(), *uname*(), *unsetenv*()
POSIX_STRING_MATCHING	*fnmatch*(), *getopt*()
POSIX_SYMBOLIC_LINKS	*lstat*(), *readlink*(), *symlink*()
POSIX_SYSTEM_DATABASE	*getgrgid*(), *getgrgid_r*(), *getgrnam*(), *getgrnam_r*(), *getpwnam*(), *getpwnam_r*(), *getpwuid*(), *getpwuid_r*()
POSIX_THREADS_BASE[f]	*pthread_atfork*(), *pthread_attr_destroy*(), *pthread_attr_getdetachstate*(), *pthread_attr_getschedparam*(), *pthread_attr_init*(), *pthread_attr_setdetachstate*(), *pthread_attr_setschedparam*(), *pthread_cancel*(), *pthread_cleanup_pop*(), *pthread_cleanup_push*(), *pthread_cond_broadcast*(), *pthread_cond_destroy*(), *pthread_cond_init*(), *pthread_cond_signal*(), *pthread_cond_timedwait*(), *pthread_cond_wait*(), *pthread_condattr_destroy*(), *pthread_condattr_init*(), *pthread_create*(), *pthread_detach*(), *pthread_equal*(), *pthread_exit*(), *pthread_getspecific*(), *pthread_join*(), *pthread_key_create*(), *pthread_key_delete*(), *pthread_kill*(), *pthread_mutex_destroy*(), *pthread_mutex_init*(), *pthread_mutex_lock*(), *pthread_mutex_trylock*(), *pthread_mutex_unlock*(), *pthread_mutexattr_destroy*(), *pthread_mutexattr_init*(), *pthread_once*(), *pthread_self*(), *pthread_setcalcelstate*(), *pthread_setcanceltype*(), *pthread_setspecific*(), *pthread_sigmask*(), *pthread_testcancel*()

Table 1-1: POSIX.1 Units of Functionality (Continued)

Unit of Functionality	Included Functions
POSIX_USER_GROUPS	*getegid*(), *geteuid*(), *getgid*(), *getgroups*(), *getlogin*(), *getlogin_r*(), *getuid*(), *setegid*(), *seteuid*, *setgid*(), *setuid*()
POSIX_WIDE_CHAR_IO	*fgetwc*(), *fgetws*(), *fputwc*(), *fputws*(), *fwide*(), *fwprintf*(), *fwscanf*(), *getwc*(), *getwchar*(), *putwc*(), *putwchar*(), *ungetwc*(), *vfwprintf*(), *vfwscanf*(), *vwprintf*(), *vwscanf*(), *wprintf*(), *wscanf*()
XSI_C_LANG_SUPPORT	*_tolower*(), *_toupper*(), *a64l*(), *daylight*(), *drand48*(), *erand48*(), *ffs*(), *getcontext*(), *getdate*(), *getsubopt*(), *hcreate*(), *hdestroy*(), *hsearch*(), *iconv*(), *iconv_close*(), *iconv_open*(), *initstate*(), *insque*(), *isascii*(), *jrand48*(), *l64a*(), *lcong48*(), *lfind*(), *lrand48*(), *lsearch*(), *makecontext*(), *memccpy*(), *mrand48*(), *nrand48*(), *random*(), *remque*(), *seed48*(), *setcontext*(), *setstate*(), *srand48*(), *srandom*(), *strcasecmp*(), *strdup*(), *strfmon*(), *strncasecmp*(), *strptime*(), *swab*(), *swapcontext*(), *tdelete*(), *tfind*(), *timezone*(), *toascii*(), *tsearch*(), *twalk*()
XSI_DBM	*dbm_clearerr*(), *dbm_close*(), *dbm_delete*(), *dbm_error*(), *dbm_fetch*(), *dbm_firstkey*(), *dbm_nextkey*(), *dbm_open*(), *dbm_store*()
XSI_DEVICE_IO	*fmtmsg*(), *poll*(), *pread*(), *pwrite*(), *readv*(), *writev*()
XSI_DEVICE_SPECIFIC	*grantpt*(), *posix_openpt*(), *ptsname*(), *unlockpt*()
XSI_DYNAMIC_LINKING	*dlclose*(), *dlerror*(), *dlopen*(), *dlsym*()
XSI_FD_MGMT	*truncate*()
XSI_FILE_SYSTEM	*basename*(), *dirname*(), *fchdir*(), *fstatvfs*(), *ftw*(), *lchown*(), *lockf*(), *mknod*(), *mkstemp*(), *nftw*(), *realpath*(), *seekdir*(), *statvfs*(), *sync*(), *telldir*(), *tempnam*()
XSI_I18N	*catclose*(), *catgets*(), *catopen*(), *nl_langinfo*()
XSI_IPC	*ftok*(), *msgctl*(), *msgget*(), *msgrcv*(), *msgsnd*(), *semctl*(), *semget*(), *semop*(), *shmat*(), *shmctl*(), *shmdt*(), *shmget*()
XSI_JOB_CONTROL	*tcgetsid*()
XSI_JUMP	*_longjmp*(), *_setjmp*()
XSI_MATH	*j0*(), *j1*(), *jn*(), *scalb*(), *y0*(), *y1*(), *yn*()
XSI_MULTI_PROCESS	*getpgid*(), *getpriority*(), *getrlimit*(), *getrusage*(), *getsid*(), *nice*(), *setpgrp*(), *setpriority*(), *setrlimit*(), *ulimit*(), *usleep*(), *vfork*(), *waitid*()
XSI_SIGNALS	*bsd_signal*(), *killpg*(), *sigaltstack*(), *sighold*(), *sigignore*(), *siginterrupt*(), *sigpause*(), *sigrelse*(), *sigset*(), *ualarm*()
XSI_SINGLE_PROCESS	*gethostid*(), *gettimeofday*(), *putenv*()
XSI_SYSTEM_DATABASE	*endpwent*(), *getpwent*(), *setpwent*()
XSI_SYSTEM_LOGGING	*closelog*(), *openlog*(), *setlogmask*(), *syslog*()

Table 1-1: POSIX.1 Units of Functionality (Continued)

Unit of Functionality	Included Functions
XSI_THREAD_MUTEX_EXT	*pthread_mutexattr_gettype*(), *pthread_mutexattr_settype*()
XSI_THREADS_EXT	*pthread_attr_getguardsize*(), *pthread_attr_getstack*(), *pthread_attr_setguardsize*(), *pthread_attr_setstack*(), *pthread_getconcurrency*(), *pthread_setconcurrency*()
XSI_TIMERS	*getitimer*(), *setitimer*()
XSI_USER_GROUPS	*endgrent*(), *endutxent*(), *getgrent*(), *getutxent*(), *getutxid*(), *getutxline*(), *pututxline*(), *setgrent*(), *setregid*(), *setreuid*(), *setutxent*()
XSI_WIDE_CHAR	*wcswidth*(), *wcwidth*()

a. There is a matching option in POSIX.1 called _POSIX_JOB_CONTROL, but that standard does not describe which functions fall under that option.
b. There is a matching option in POSIX.1 called _POSIX_REGEXP, but that standard does not describe which functions fall under that option.
c. There is a matching option in POSIX.1 called _POSIX_READER_WRITER_LOCKS, but that standard does not describe which functions fall under that option.
d. Dependent on the _POSIX_TIMEOUTS option.
e. Dependent on the _POSIX_THREAD_PROCESS_SHARED option.
f. POSIX_THREADS_BASE is the same as the _POSIX_THREADS option, but without the functions belonging to the POSIX_RW_LOCKS unit of functionality.

Table 1-2: POSIX.5 Units of Functionality (Ada Language Support)

POSIX_ADA_LANG_SUPPORT	
Package	**Subprograms**
`System`	Extra requirements specified in POSIX.5c, Section 2.8
`System_Storage_Elements`	All[a]
`POSIX_Page_Alignment`	All
`POSIX_Supplement_To_Ada_IO`	All
`Ada_Task_Identification`	All
`Ada_Streams`	All

a. *All*: indicates all subprograms in a package are required to be supported. Where overloaded versions of a subprogram exist, each instance is required, except as noted. All `Image` and `Value` functions must be supported for all packages provided by the implementation.

Table 1-3: POSIX.5 Units of Functionality (Device IO)

POSIX_DEVICE_IO	
Package	**Subprograms**
POSIX_IO	Open Close Read Write Generic_Read Generic_Write Is_Open

Table 1-4: POSIX.5 Units of Functionality (Device Specific)

POSIX_DEVICE_SPECIFIC	
Package	**Subprograms**
POSIX_Terminal_Functions	Get_Terminal_Characteristics Get_Controlling_Terminal_Name Set_Terminal_Characteristics Terminal_Modes_Of Define_Terminal_Modes Bits_Per_Character_Of Define_Bits_Per_Character Special_Control_Character_Of Define_Special_Control_Character Disable_Control_Character Input_Time_Of Define_Input_Time Minimum_Input_Count_Of Define_Minimum_Input_Count Input_Baud_Rate_Of Output_Baud_Rate_Of Define_Input_Baud_Rate Define_Output_Baud_Rate Send_Break Drain Discard_Data Flow
POSIX_IO	Is_A_Terminal Get_Terminal_Name

Table 1-5: POSIX.5 Units of Functionality (Event Management)

POSIX_EVENT_MGMT	
Package	**Subprograms**
POSIX_Event_Management[a]	Make_Empty Add Remove In_Set Select_File For_Every_File_In

a. The subprograms listed in this table are those under the Select option in POSIX.5c. But instead of using this option, a unit of functionality has been created because there is no equivalent option in POSIX.1.

Table 1-6: POSIX.5 Units of Functionality (FD Management)

POSIX_FD_MGMT	
Package	**Subprograms**
POSIX_File_Locking	All
POSIX_IO	Duplicate Duplicate_And_Close Get_File_Control Set_File_Control Get_Close_On_Exec Set_Close_On_Exec Seek File_Size File_Position

Table 1-7: POSIX.5 Units of Functionality (FIFO)

POSIX_FIFO	
Package	**Subprograms**
POSIX_Files	Create_FIFO

Table 1-8: POSIX.5 Units of Functionality (File Attributes)

POSIX_FILE_ATTRIBUTES	
Package	**Subprograms**
POSIX_Permissions	Set_Allowed_Process_Permissions Get_Allowed_Process_Permissions
POSIX_Files	Change_Owner_And_Group Change_Permissions

Table 1-9: POSIX.5 Units of Functionality (File System)

POSIX_FILE_SYSTEM	
Package	**Subprograms**
POSIX_Configurable_File_Limits	All
POSIX_File_Status	All

Table 1-9: POSIX.5 Units of Functionality (File System) (Continued)

POSIX_FILE_SYSTEM	
Package	**Subprograms**
POSIX_Files	For_Every_Directory_Entry Create_Directory Unlink Remove_Directory Rename Accessibility Is_Accessible Existence Is_File_Present Set_File_Times Link Filename_Of Is_File Is_Directory Is_FIFO Is_Character_Special_File Is_Block_Special_File Is_Socket
POSIX_Process_Environment	Change_Working_Directory Get_Working_Directory
POSIX_IO	Open_Or_Create

Table 1-10: POSIX.5 Units of Functionality (Job Control)

POSIX_JOB_CONTROL[a]	
Package	**Subprograms**
POSIX_Process_Identification	Set_Process_Group_Id Create_Process_Group
POSIX_Terminal_Functions	Get_Process_Group_Id Set_Process_Group_Id
POSIX_Signals	Set_Stopped_Child_Signal Stopped_Child_Signal_Enabled

a. The subprograms listed in this table are those under the Job Control option in POSIX.5c. But instead of using this option, a unit of functionality has been created because the equivalent option in POSIX.1 does not specify the functions that fall under it.

Table 1-11: POSIX.5 Units of Functionality (Multi-Process)

POSIX_MULTI_PROCESS	
Package	**Subprograms**
POSIX_Process_Primitives	All
POSIX_Unsafe_Process_Primitives	All
POSIX_Process_Times	All
POSIX_Process_Identification	Get_Process_Id Get_Parent_Process_Id

Table 1-12: POSIX.5 Units of Functionality (Networking)

POSIX_NETWORKING	
Package	**Subprograms**

`POSIX_IO`	`Get_Owner`
	`Set_Socket_Process_Owner`
	`Set_Socket_Group_Owner`
	`Set_Buffer`
	`Get_Buffer`

`POSIX_Sockets`	All[a]
`POSIX_Sockets_Local`	All[a]
`POSIX_Sockets_Internet`	All[b]

a. The `POSIX_Sockets` and `POSIX_Sockets_Local` packages depend on the Sockets Detailed Network Interface option (and partly on the Network Management option) defined in POSIX.5c, but they are included here because there are no equivalent options in POSIX.1.

b. The `POSIX_Sockets_Internet` package depends on the Sockets Detailed Network Interface option (and partly on the Internet Protocol, Internet Datagram, and Internet Stream options) defined in POSIX.5c, but it is included here because there are no equivalent options in POSIX.1.

Table 1-13: POSIX.5 Units of Functionality (Pipes)

POSIX_PIPES	
Package	**Subprograms**
`POSIX_IO`	`Create_Pipe`

Table 1-14: POSIX.5 Units of Functionality (Priority Ranges)

POSIX_PRIORITY_RANGES	
Package	**Subprograms**

`POSIX_Process_Scheduling`	`Get_Maximum_Priority`
	`Get_Minimum_Priority`
	`Get_Round_Robin_Interval`

Table 1-15: POSIX.5 Units of Functionality (Signals)

POSIX_SIGNALS	
Package	**Subprograms**
POSIX_Signals	Add_Signal Add_All_Signals Delete_Signal Delete_All_Signals Is_Member Send_Signal Set_Blocked_Signals Block_Signals Unblock_Signals Blocked_Signals Ignore_Signal Unignore_Signal Is_Ignored Install_Empty_Handler Pending_Signals Await_Signal[a] Await_Signal_Or_Timeout[a] Interrupt_Task Get_Signal[b] Set_Signal[b] Get_Notification Set_Notification Get_Data[b] Set_Data[b]

a. Return type Signal
b. Operation on type Signal_Event

Table 1-16: POSIX.5 Units of Functionality (Single Process)

POSIX_SINGLE_PROCESS	
Package	**Subprograms**
POSIX	All
POSIX_Limits	All
POSIX_Options	All
POSIX_Profiles	All[a]
POSIX_Configurable_System_Limits	All
POSIX_Calendar	All

Table 1-16: POSIX.5 Units of Functionality (Single Process) (Continued)

POSIX_SINGLE_PROCESS	
Package	**Subprograms**
POSIX_Process_Environment	Argument_List Copy_From_Current_Environment Copy_To_Current_Environment Copy_Environment Clear_Environment Set_Environment_Variable Delete_Environment_Variable Length For_Every_Environment_Variable For_Every_Current_Environment_Variable Environment_Value_Of Is_Environment_Variable

a. The POSIX_Profiles package is defined in Annex B of this document

Table 1-17: POSIX.5 Units of Functionality (System Database)

POSIX_SYSTEM_DATABASE	
Package	**Subprograms**
POSIX_Group_Database	All
POSIX_User_Database	All

Table 1-18: POSIX.5 Units of Functionality (User Groups)

POSIX_USER_GROUPS	
Package	**Subprograms**
POSIX_Process_Identification	Get_Real_User_ID Get_Effective_User_ID Get_Real_Group_ID Get_Effective_Group_ID Set_User_ID Create_Session Set_Group_ID Get_Groups Get_Login_Name Get_Process_Group_ID

1.5 Development Environment

Although the Shell and Utilities part of POSIX.1 is not required for the execution environment of PSE51, PSE52, or PSE53, option POSIX2_SW_DEV is required in the development environments for all four profiles. The options POSIX2_C_BIND and POSIX2_C_DEV are required for C-Language development environments.

1.6 Summary of Profile Features

The following tables summarize the requirements of the four profiles. Since POSIX.1, and/or POSIX.5 do not provide sufficient options to remove features unnecessary for some profiles, Units of Functionality have been developed and are described in Table 1-1 through Table 1-18 and defined by Annex A and Annex B.

Table 1-19: Units of Functionality Requirements

Unit of Functionality	PSE51	PSE52	PSE53	PSE54
POSIX_ADA_LANG_SUPPORT[a]	X	X	X	X
POSIX_C_LANG_JUMP[b]	X	X	X	X
POSIX_C_LANG_MATH[b]	-	X	X	X
POSIX_C_LANG_SUPPORT[b]	X	X	X	X
POSIX_C_LANG_WIDE_CHAR[b]	-	-	-	X
POSIX_DEVICE_IO	X	X	X	X
POSIX_DEVICE_SPECIFIC	-	-	-	X
POSIX_EVENT_MGMT	-	-	X	X
POSIX_FD_MGMT	-	X	X	X
POSIX_FIFO	-	-	-	X
POSIX_FILE_ATTRIBUTES	-	-	-	X
POSIX_FILE_LOCKING[b]	X	X	X	X
POSIX_FILE_SYSTEM	-	X	X	X
POSIX_FILE_SYSTEM_EXT[b]	-	-	-	X
POSIX_JOB_CONTROL	-	-	-	X
POSIX_MULTI_PROCESS	-	-	X	X
POSIX_NETWORKING	-	-	X	X
POSIX_PIPE	-	-	X	X
POSIX_PRIORITY_RANGES	X	X	-	-
POSIX_REGEXP[b]	-	-	-	X
POSIX_RW_LOCKS[b]	-	-	-	-
POSIX_SHELL_FUNC[b]	-	-	-	X
POSIX_SIGNALS	X	X	X	X
POSIX_SIGNAL_JUMP[b]	-	-	X	X
POSIX_SINGLE_PROCESS	X	X	X	X
POSIX_STRING_MATCHING[b]	-	-	-	X
POSIX_SYMBOLIC_LINKS[b]	-	-	-	X
POSIX_SYSTEM_DATABASE	-	-	-	X
POSIX_THREADS_BASE[b]	X	X	X	X
POSIX_USER_GROUPS	-	-	-	X
POSIX_WIDE_CHAR_IO[b]	-	-	-	X
XSI_C_LANG_SUPPORT[b]	-	-	-	-
XSI_DBM[b]	-	-	-	-

Table 1-19: Units of Functionality Requirements (Continued)

Unit of Functionality	PSE51	PSE52	PSE53	PSE54
XSI_DEVICE_IO[b]	-	-	-	-
XSI_DEVICE_SPECIFIC[b]	-	-	-	-
XSI_DYNAMIC_LINKING[b]	-	-	-	X
XSI_FD_MGMT[b]	-	-	-	-
XSI_FILE_SYSTEM[b]	-	-	-	-
XSI_I18N[b]	-	-	-	-
XSI_IPC[b]	-	-	-	-
XSI_JOB_CONTROL[b]	-	-	-	-
XSI_JUMP[b]	-	-	-	-
XSI_MATH[b]	-	-	-	-
XSI_MULTI_PROCESS[b]	-	-	-	-
XSI_SIGNALS[b]	-	-	-	-
XSI_SINGLE_PROCESS[b]	-	-	-	-
XSI_SYSTEM_DATABASE[b]	-	-	-	-
XSI_SYSTEM_LOGGING[b]	-	-	-	X
XSI_THREAD_MUTEX_EXT[b]	X	X	X	X
XSI_THREADS_EXT[b]	X	X	X	X
XSI_TIMERS[b]	-	-	-	-
XSI_USER_GROUPS[b]	-	-	-	-
XSI_WIDE_CHAR[b]	-	-	-	-

a. Required only for the Ada-Language option
b. Required only for the C-Language option

Table 1-20: POSIX.1 Option Requirements

Option	PSE51	PSE52	PSE53	PSE54
_POSIX_ADVISORY_INFO	-	-	-	X
_POSIX_ASYNCHRONOUS_IO	-	-	X	X
_POSIX_BARRIERS	-	-	-	-
_POSIX_CHOWN_RESTRICTED	-	-	-	X
_POSIX_CLOCK_SELECTION	X	X	X	X
_POSIX_CPUTIME	-	-	X	X
_POSIX_FSYNC	X	X	X	X
_POSIX_IPV6	-	-	-	-
_POSIX_JOB_CONTROL	-	-	-	X
_POSIX_MAPPED_FILES	-	X	X	X
_POSIX_MEMLOCK	X	X	X	X
_POSIX_MEMLOCK_RANGE	X	X	X	X
_POSIX_MEMORY_PROTECTION	-	-	X	X
_POSIX_MESSAGE_PASSING	-	X	X	X

Table 1-20: POSIX.1 Option Requirements (Continued)

Option	PSE51	PSE52	PSE53	PSE54
_POSIX_MONOTONIC_CLOCK	X	X	X	X
_POSIX_NO_TRUNC	X	X	X	X
_POSIX_PRIORITIZED_IO	-	-	X	X
_POSIX_PRIORITY_SCHEDULING	-	-	X	X
_POSIX_RAW_SOCKETS	-	-	X	X
_POSIX_READER_WRITER_LOCKS	X	X	X	X
_POSIX_REALTIME_SIGNALS	X	X	X	X
_POSIX_REGEXP	-	-	-	X
_POSIX_SAVED_IDS	-	-	-	X
_POSIX_SEMAPHORES	X	X	X	X
_POSIX_SHARED_MEMORY_OBJECTS	X	X	X	X
_POSIX_SHELL	-	-	-	X
_POSIX_SPAWN	-	-	X	X
_POSIX_SPIN_LOCKS	-	-	-	-
_POSIX_SPORADIC_SERVER	-	-	X	X
_POSIX_SYNCHRONIZED_IO	X	X	X	X
_POSIX_THREAD_ATTR_STACKADDR	X	X	X	X
_POSIX_THREAD_ATTR_STACKSIZE	X	X	X	X
_POSIX_THREAD_CPUTIME	X	X	X	X
_POSIX_THREAD_PRIO_INHERIT	X	X	X	X
_POSIX_THREAD_PRIO_PROTECT	X	X	X	X
_POSIX_THREAD_PRIORITY_SCHEDULING	X	X	X	X
_POSIX_THREAD_PROCESS_SHARED	-	-	X	X
_POSIX_THREAD_SAFE_FUNCTIONS	See units of functionality			
_POSIX_THREAD_SPORADIC_SERVER	X	X	X	X
_POSIX_THREADS	See units of functionality			
_POSIX_TIMEOUTS	X	X	X	X
_POSIX_TIMERS	X	X	X	X
_POSIX_TRACE	-	X	X	X
_POSIX_TRACE_EVENT_FILTER	-	X	X	X
_POSIX_TRACE_INHERIT	-	-	-	-
_POSIX_TRACE_LOG	-	X	X	X
_POSIX_TYPED_MEMORY_OBJECTS	-	-	-	-
_POSIX_VDISABLE	-	-	-	X
_POSIX2_C_BIND[a]	X[b]	X[b]	X[b]	X
_POSIX2_C_DEV[a]	X[b]	X[b]	X[b]	X
_POSIX2_CHAR_TERM	-	-	-	X
_POSIX2_FORT_DEV	-	-	-	-
_POSIX2_FORT_RUN	-	-	-	X
_POSIX2_LOCALEDEF	-	-	-	-
_POSIX2_PBS	-	-	-	-
_POSIX2_PBS_ACCOUNTING	-	-	-	-

Table 1-20: POSIX.1 Option Requirements (Continued)

Option	PSE51	PSE52	PSE53	PSE54
_POSIX2_PBS_CHECKPOINT	-	-	-	-
_POSIX2_PBS_LOCATE	-	-	-	-
_POSIX2_PBS_MESSAGE	-	-	-	-
_POSIX2_PBS_TRACK	-	-	-	-
_POSIX2_SW_DEV	X[b]	X[b]	X[b]	X
_POSIX2_UPE	-	-	-	X
_XOPEN_CRYPT	-	-	-	-
_XOPEN_ENH_I18N	No interfaces fall under this option			
_XOPEN_LEGACY	-	-	-	-
_XOPEN_REALTIME	See individual suboptions			
_XOPEN_REALTIME_THREADS	See individual suboptions			
_XOPEN_SHM	No interfaces fall under this option			
_XOPEN_STREAMS	-	-	-	-
_XOPEN_UNIX	See units of functionality			

a. Required only for the C-language option.
b. Required only for the development platform, which will often differ from the execution platform.

The correspondence between the options listed in Table 1-20 and the options described in POSIX.5c, clause 2.5, are as follows:

Table 1-21: POSIX.1 Options vs. POSIX.5c Options

POSIX.1 Option	POSIX.5c Option
_POSIX_ADVISORY_INFO	none
_POSIX_ASYNCHRONOUS_IO	**Asynchronous I/O**
_POSIX_BARRIERS	none
_POSIX_CHOWN_RESTRICTED	**Change Owner Restriction**
_POSIX_CLOCK_SELECTION	none
_POSIX_CPUTIME	none
_POSIX_FSYNC	**File Synchronization**
_POSIX_IPV6	none
_POSIX_MAPPED_FILES	**Memory Mapped Files**
_POSIX_MEMLOCK	**Memory Locking**
_POSIX_MEMLOCK_RANGE	**Memory Range Locking**
_POSIX_MEMORY_PROTECTION	**Memory Protection**
_POSIX_MESSAGE_PASSING	**Message Queues**
_POSIX_MONOTONIC_CLOCK	none
_POSIX_NO_TRUNC	**Filename Truncation**[a]
_POSIX_PRIORITIZED_IO	**Prioritized I/O**
_POSIX_PRIORITY_SCHEDULING	**Priority Process Scheduling**
_POSIX_RAW_SOCKETS	none
_POSIX_READER_WRITER_LOCKS	none
_POSIX_REALTIME_SIGNALS	**Realtime Signals**

Table 1-21: POSIX.1 Options vs. POSIX.5c Options (Continued)

POSIX.1 Option	POSIX.5c Option
_POSIX_REGEXP	none
_POSIX_SAVED_IDS	**Saved IDs Support**
_POSIX_SEMAPHORES	**Semaphores**
_POSIX_SHARED_MEMORY_OBJECTS	**Shared Memory Objects**
_POSIX_SHELL	not applicable
_POSIX_SPAWN	C-language specific
_POSIX_SPIN_LOCKS	none
_POSIX_SPORADIC_SERVER	none
_POSIX_SYNCHRONIZED_IO	**Synchronized I/O**
_POSIX_THREAD_ATTR_STACKADDR	C-language specific
_POSIX_THREAD_ATTR_STACKSIZE	C-language specific
_POSIX_THREAD_CPUTIME	none
_POSIX_THREAD_PRIO_INHERIT	**Mutex Priority Inheritance**
_POSIX_THREAD_PRIO_PROTECT	**Mutex Priority Ceiling**
_POSIX_THREAD_PRIORITY_SCHEDULING	C-language specific
_POSIX_THREAD_PROCESS_SHARED	**Process Shared**
_POSIX_THREAD_SAFE_FUNCTIONS	C-language specific
_POSIX_THREAD_SPORADIC_SERVER	none
_POSIX_THREADS	C-language specific
_POSIX_TIMEOUTS	none
_POSIX_TIMERS	**Timers**
_POSIX_TRACE	none
_POSIX_TRACE_EVENT_FILTER	none
_POSIX_TRACE_INHERIT	none
_POSIX_TRACE_LOG	none
_POSIX_TYPED_MEMORY_OBJECTS	none
_POSIX_VDISABLE	C-language specific
_POSIX2_C_BIND	not applicable
_POSIX2_C_DEV	not applicable
_POSIX2_CHAR_TERM	not applicable
_POSIX2_FORT_DEV	not applicable
_POSIX2_FORT_RUN	not applicable
_POSIX2_LOCALEDEF	not applicable
_POSIX2_PBS	not applicable
_POSIX2_PBS_ACCOUNTING	not applicable
_POSIX2_PBS_CHECKPOINT	not applicable
_POSIX2_PBS_LOCATE	not applicable
_POSIX2_PBS_MESSAGE	not applicable
_POSIX2_PBS_TRACK	not applicable
_POSIX2_SW_DEV	not applicable
_POSIX2_UPE	not applicable
_XOPEN_CRYPT	none
_XOPEN_ENH_I18N	none

Table 1-21: POSIX.1 Options vs. POSIX.5c Options (Continued)

POSIX.1 Option	POSIX.5c Option
_XOPEN_LEGACY	none
_XOPEN_REALTIME	none
_XOPEN_REALTIME_THREADS	none
_XOPEN_SHM	none
_XOPEN_STREAMS	none
_XOPEN_UNIX	none

a. Note that the POSIX.5c Filename Truncation option has the opposite sense relative to the POSIX.1 option _POSIX_NO_TRUNC

In all profiles that do not support the POSIX_JOB_CONTROL unit of functionality, the subprogram `POSIX_Signals.Set_Stopped_Child_Signal` shall fail silently.

In all profiles that do not support the POSIX_JOB_CONTROL unit of functionality, the subprogram `POSIX_Signals.Stopped_Child_Signal_Enabled` shall return `False`.

`POSIX_Limits.Groups_Maxima'First` shall be zero for PSE51, PSE52, and PSE53. For PSE54 it shall be greater than or equal to eight.

`POSIX_Terminal_Functions.Disable_Control_Character` (which corresponds to _POSIX_VDISABLE is not supported in PSE51, PSE52, and PSE53. For PSE54, `POSIX_Terminal_Functions.Disable_Control_Character` shall not raise `POSIX_Error` with an error code of `Operation_Not_Implemented`.

For PSE51 and PSE52, the blocking behavior of all reentrant operations defined by POSIX.5c shall be per task, i.e., a blocked task cannot prevent any other task from executing. Therefore, the corresponding `Blocking_Behavior` constants shall have the value `Tasks`. (See POSIX.5c, clause 2.4.1.5.)

Section 2: Normative References

2.1 Normative References

The following standards contain provisions which, through references in this text, constitute provisions of this standard[1]. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this profile of IEEE and ISO are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

{1} ISO/IEC 8652:1995[2], *Information Technology—Programming languages—Ada*, including Technical Corrigendum No.1.

{2} ISO/IEC 9899:1999, *Information processing systems—Programming languages—C,* including Technical Corrigendum No. 1.

{3} IEEE Std 1003.1:2001, *Information Technology—Portable Operating System Interface* (*POSIX*®) (Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992)[3].

{4} IEEE Std 1003.5-1992, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API)*.

{5} IEEE Std 1003.5b-1996, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API)—Amendment 1: Realtime Extensions*.

1. Other references to related standards and other documents can be found in Annex C of this document. Common names for these standards can be found in 4.2, “Abbreviations”.
2. ISO/IEC documents can be obtained from the ISO office, 1 rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/) and from the IEC office, 3 rue de Varembé, Case Postale 131, CH-1211, Genève 20, Switzerland/Suisse (http://www.iec.ch/). ISO/IEC publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (http://www.ansi.org/).
3. Includes IEEE Std 1003.1d-1999, IEEE Std 1003.1j-1999, and IEEE Std 1003.1q-2000

{6} IEEE Std 1003.5c-1998, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API)—Amendment 2: Protocol Independent Interfaces*.

{7} ISO/IEC TR 10000-1:1998 *Information technology -- Framework and taxonomy of International Standardized Profiles -- Part 1: General principles and documentation framework*.

{8} ISO/IEC TR 10000-3:1998 *Information technology -- Framework and taxonomy of International Standardized Profiles -- Part 3: Principles and Taxonomy for Open System Environment Profiles*.

Section 3: Terms and Definitions

3.1 Terminology

For the purposes of this standard, the following terms apply:

3.1.1 implementation defined: Describes a value or behavior that is not defined by the standard, but is selected by an implementor. The value or behavior may vary among implementations that conform to POSIX.13. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior in the conformance document, so that it can be used correctly by an application.

3.1.2 may: Describes a feature or behavior that is optional for an implementation that conforms to POSIX.13. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

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

3.1.3 shall: For an implementation that conforms to POSIX.13, describes a feature or behavior that is mandatory. An application can rely on the existence of the feature or behavior.

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

3.1.4 should: For an implementation that conforms to POSIX.13, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

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

3.1.5 undefined: Describes the nature of a value or behavior not defined by POSIX.13 which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to POSIX.13. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

3.1.6 unspecified: Describes the nature of a value or behavior not specified by POSIX.13 which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to POSIX.13. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

3.2 Definitions

For the purposes of this standard, the following definitions apply:

3.2.1 Application Environment Profile (AEP): An OSE profile which specifies a complete and coherent subset of the Open System Environment. [ISO/IEC TR 10000-3:1998 {8}]

3.2.2 Application Platform: A set of resources on which an application will run.

3.2.3 Base Standard: An approved IEEE, National, Regional, or International Standard which defines and describes basic functionality and capability. [ISO/IEC TR 10000-1:1998 {7}]

3.2.4 Component Profile: An Application Environment Profile that specifies a unit of functionality in terms of the interfaces that it supports and the interfaces

that it uses, and the relationships between these interfaces. [ISO/IEC TR 10000-3:1998 {8}]

3.2.5 Conformance Document: A document provided by an implementor that contains implementation details as described in 5.1.1.2.

3.2.6 Development Platform: A system used to prepare an application for execution. Such a system is possibly distinct from the system on which the application will execute.

3.2.7 Embedded Computer System: A computer (and its software) is considered *embedded* if it is an integral component of a larger system and is used to control and/or directly monitor that system, using special hardware devices.

3.2.8 Generic Application Environment Profile: An Application Environment Profile which is not specific to a particular community of use [ISO/IEC TR 10000-3:1998 {8}]

3.2.9 Generic Interface Profile: An Interface Profile which is not specific to a particular community of use. [ISO/IEC TR 10000-3:1998 {8}]

3.2.10 Industry Specific Application Environment Profile: An Application Environment Profile which deals with specific industry requirements. [ISO/IEC TR 10000-3:1998 {8}]

3.2.11 Industry Specific Interface Profile: An Interface Profile which deals with specific industry requirements. [ISO/IEC TR 10000-3:1998 {8}]

3.2.12 Interface Profile (IP): An OSE Profile defining one interface of the Open System Environment. [ISO/IEC TR 10000-3:1998 {8}]

3.2.13 International Standardized Profile (ISP): An internationally agreed-to, harmonized document which identifies a standard or group of standards, together with options and parameters, necessary to accomplish a function or set of functions. [ISO/IEC TR 10000-1:1998 {8}]

3.2.14 Open System Environment (OSE): The comprehensive set of interfaces, services, and supporting formats for interoperability and/or for portability of applications, data or people, as specified by information technology standards and profiles. [ISO/IEC TR 10000-3:1998 {8}]

3.2.15 Priority Inversion: A condition in which a thread that is waiting for a shared resource (including a CPU) is prevented from executing while a thread with a lower application-specified priority is running. The delays caused by priority inversion can be extremely large in the case of unbounded priority inversion (see definition). But there are mechanisms to bound these delays to small predictable intervals.

See also 3.2.21, “Unbounded Priority Inversion”.

3.2.16 Profile (for ISO standardization): A set of one or more base standards (and where applicable) chosen classes, subsets, options, and parameters of those base standards to accomplish a function. [ISO/IEC TR 10000-1:1998 {7}]

3.2.17 Realtime Environment Profile: A profile designed to support applications requiring bounded response.

3.2.18 System Documentation: All documentation provided with an implementation, except the conformance document.

Electronically distributed documents for an implementation are considered part of the system documentation.

3.2.19 Subprofiling Option Group: A unit of functionality (See 3.2.22).

3.2.20 System Profile: An Application Environment Profile that specifies a set of functions necessary to support a class of applications. It specifies the behavior to be observed at the interfaces of the application platform on which the class of applications can run. [ISO/IEC TR 10000-3:1998 {8}]

NOTE: A system profile is defined in terms of component profiles that specify units of functionality that can be combined to realize the application platform.

3.2.21 Unbounded Priority Inversion: A priority inversion condition in which the delay caused to the waiting thread cannot be bounded by the duration of the intervals during which lower priority threads hold the shared resource. For exam-

ple, this can happen when a lower priority thread is holding a lock also requested by the high priority thread, and then one or more medium priority threads request execution, thus preempting the lower priority thread

See also 3.2.15, “Priority Inversion”.

3.2.22 Unit of Functionality: A separately implementable element of an OSE system. [ISO/IEC TR 10000-3:1998 {8}]

3.3 Rationale for definitions

(This subclause is not a normative part of IEEE Std P1003.13)

Embedded Computer System. For the definition of an embedded computer system the following canonical examples were taken into account:

— Are programs that understand physics and/or hardware embedded? For example one that uses finite-element methods to predict fluid flow over airplane wings? No. These programs are never considered to be embedded because they are not an integral component of a larger system.

— Is the internal microprocessor controlling a disk drive an example of an embedded system? Yes, regardless of what the disk drive is used for. The software (firmware, actually) within the disk drive controls the HDA (head disk assembly) hardware, and is hard realtime as well.

— I/O drivers control hardware, so does presence of an I/O driver imply that the computer executing the driver is embedded? No, because that computer may be a general-purpose computer that is not part of a larger system.

— Is a PDA (Personal Digital Assistant) an embedded system? No. People often say that PDAs are embedded because they are very small and constrained, and because PDA OS and application software is kept in nonvolatile memory, but PDAs parallel the desktop systems used to run office productivity applications, and no special hardware is being controlled.

— Is the microprocessor controlling a cellphone an embedded system? Yes. The firmware in the cellphone is controlling the radio hardware.

— Are the computers in a big phased-array radar considered embedded? These radars are ten-story buildings with one to three 100-foot diameter radiating patches on the sloped sides of the building. Yes. These computers are generally some of the most powerful computers available when the system was built, live in a large computer room occupying almost one whole floor of a building, may be hundreds of meters away from the radar hard-

ware, but these computers are still an integral component of a larger system.

— Is a traditional Flight Management System (FMS) built into an airplane cockpit considered embedded? If the FMS is not connected to the avionics, and is used only for logistics computations, a function readily performed on a laptop, then the FMS is clearly not embedded.

— Are the computers in a hardware-in-the-loop (HIL) simulator embedded? Yes, both in the simulator, and in the thing being tested in the HIL simulator. Hardware is being controlled on both sides.

— Is the computer controlling a pacemaker in a person's chest an embedded computer? Yes. In this case the "system" is the combination of the pacemaker and the person's heart.

— Is the computer controlling fuel injection in an automobile engine embedded? Yes. It is part of a larger system, the engine, and it is directly monitoring and controlling the engine through special hardware.

Section 4: Conventions and Abbreviations

4.1 Conventions

This document uses the following typographic conventions:

(1) The *italic* font is used for:

— Symbolic parameters that are generally substituted with real values by the application

— C language data types and function names

— Global external variable names

— Function families; references to groups of closely related functions

(2) The bold font is used with a word in all capital letters, such as

PATH

to represent an environment variable. It is also used for the term "**NULL** pointer."

Sometimes it is used in tables to enhance visibility of option names.

(3) The `constant-width` (Courier) font is used:

— For references to utility names and C language headers

— For names of attributes in attributes objects

— For references to Ada identifiers.

(4) Symbolic constants returned by many functions as error numbers are represented as:

[ERRNO]

(5) Symbolic constants or limits defined in certain headers are represented as:

_POSIX_AEP_REALTIME_

In some cases tabular information is presented "inline"; in others it is presented in a separately labeled table. This arrangement was employed purely for ease of typesetting and there is no normative difference between these two cases.

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

Notes provided as parts of labeled tables and figures are integral parts of this Standard (normative). Footnotes and notes within the body of the text are for information only (informative).

Numerical quantities are presented in international style: comma is used as a decimal sign and units are from the International System (SI).

4.2 Abbreviations

For the purposes of this document the following abbreviations apply:

4.2.1 Ada95 RM: ISO/IEC 8652:1995, *Information Technology—Programming languages—Ada [Revision of the first edition (ISO 8652:1987)]*, including Technical Corrigendum No. 1.

4.2.2 C99 Standard: ISO/IEC 9899:1999, *Information processing systems—Programming languages—C,* including Technical Corrigendum No. 1.

4.2.3 MMU: Memory management unit.

4.2.4 POSIX.1: IEEE Std 1003.1:2001, *Information Technology—Portable Operating System Interface* (*POSIX*®) (Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992).

4.2.5 POSIX.5c: IEEE Std 1003.5-1992, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API)* as amended by IEEE Std 1003.5b-1996, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API)—Amendment 1: Realtime Extensions* and IEEE Std 1003.5c-1998, *IEEE Standard for Information Technology—POSIX Ada Language Interfaces—Part 1: Binding for System Application Program Interface (API)—Amendment 2: Protocol Independent Interfaces*.

4.2.6 POSIX.13: This standard.

4.2.7 AEP: Application Environment Profile.

4.2.8 ISP: International Standardized Profile.

4.2.9 OSE: Open System Environment.

4.2.10 PSE: Generic Environment Profile.

4.2.11 PSE51: The Minimal Realtime System Profile defined herein.

4.2.12 PSE52: The Realtime Controller System Profile defined herein.

4.2.13 PSE53: The Dedicated Realtime System Profile defined herein.

4.2.14 PSE54: The Multi-Purpose Realtime System Profile defined herein.

4.2.15 PSE5X: Any one of the PSE51, PSE52, PSE53 or PSE54 profiles.

Section 5: Conformance

5.1 Conformance

5.1.1 Implementation Conformance

5.1.1.1 Requirements

An implementation may claim conformance to one or more of the profiles defined by this standard. For any given profile a conforming implementation shall meet all of the following criteria:

(1) The system shall support all required interfaces referenced in the appropriate standardized profile. These interfaces shall support the functional behavior described in the appropriate base standard, and any additional constraints or options described herein.

(2) The system may provide additional functions or facilities not required by this standard. Nonstandard extensions should be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of functions or facilities defined in the appropriate base standard. The conformance document shall define an environment in which an application can be run with predictable behavior specified by the referenced standards. In no case shall such an environment require modification of a Strictly Conforming POSIX.13 Application.

5.1.1.2 Documentation

An implementation conforming to one or more of the profiles defined by this standard shall provide a conformance document that shall document conformance in one of two specific manners:

(1) If the implementation is fully conformant to the referenced base standard(s), then that implementation may cite the separate conformance doc-

uments that document the base standard conformance. This will primarily apply to implementations that support the PSE53 or PSE54 Profiles.

(2) If the implementation does not fully conform to one or more of the referenced base standards, or if separate base standard conformance documents are not cited, the implementation shall document the specific extent of conformance to each such base standard. This specification shall include:

— A complete list of interfaces from the base standard that are present in the implementation.

— Limit values whose specification is normally required in a conformance document for the base standard (e.g. the limit values found in the <limits.h> and <unistd.h> headers), stating values, the conditions under which those values may change, and the limits of such variations, if any.

— A description of the behavior of the implementation for all implementation-defined features specified by those portions of the base standard that the implementation provides. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or providing full syntax and semantics of these features. The conformance document may specify the behavior of the implementation for those features where the referenced standards state that the implementations may vary or where features are identified as undefined or unspecified.

Regardless of whether separate base standard conformance documents are cited, the conformance document for these profile(s) shall contain a statement that indicates the full name, number, and date of the standard (i.e. the profile standard) that applies. The conformance document may also list international standards that are available for use by a Conforming POSIX.13 Application. Applicable characteristics where documentation is required by one of these standards or by standards of government bodies, may also be included.

5.1.2 Application Conformance

An application claiming conformance to one or more of these profiles shall use only the facilities described in that profile and included referenced standard elements, and shall fall within one of the categories in 5.1.2.1, 5.1.2.2, or 5.1.2.3.

Any application that conforms to one or more of these profiles under the C-Language option also conforms to POSIX.1. Any application that conforms to one or more of these profiles under the Ada-Language option also conforms to POSIX.5c.

5.1.2.1 Strictly Conforming Application

An application is said to be strictly conforming to a given POSIX.13 profile if the application requires only the facilities required in that profile. Such an application shall accept any behavior described in the profile as *unspecified* or *implementation-defined*, and for symbolic constants, shall accept any value in the range permitted by the profile. Such applications are permitted to adapt to the availability of facilities whose availability is indicated by the constants in 6.1.3, 7.1.3, 8.1.3 and 9.1.3.

5.1.2.2 Conformant Application

5.1.2.2.1 ISO/IEC Conformant Application

An application is said to be ISO/IEC Conformant to a given POSIX.13 profile if the application requires only the facilities required in that profile and approved Conformant Language bindings for any ISO or IEC standard. Such an application shall include a statement of conformance that documents all options and limit dependencies, and all other ISO or IEC standards used.

5.1.2.2.2 <National Body> Conformant POSIX.13 Application

An application is said to be <National Body> Conformant to a given POSIX.13 profile if the application requires only the facilities required in that profile. Such an application shall include a statement of conformance to document all options and limit dependencies, and all other <National Body> standards used.

5.1.2.3 Conformant Application Using Extensions

An application is said to be conformant using extensions if it only uses nonstandard facilities consistent with this standard. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Conformant Application. A Conformant Application Using Extensions shall be either an ISO/IEC Conformant Application Using Extensions or a <National Body> Conformant Application Using Extensions. (See 5.1.2.2.1 and 5.1.2.2.2)

Section 6: Minimal Realtime System Profile (PSE51)

6.1 Introduction

This section specifies those standards required for conformance to the Minimal Realtime System Profile option and, where applicable, the state of any options contained in those standards.

When a referenced standard specifies services beyond those required by the Minimal Realtime System Profile, only those services included in the specified Units of Functionality referenced by this profile shall be required (See Table 1-1 through Table 1-18). All the applicable definitions in POSIX.1 and/or POSIX.5c shall still apply.

6.1.1 Identification

For the C-Language implementation, symbolic names shall be used to specify the presence or absence of each option in this profile. Names reserved for use in this profile begin with the string _POSIX_AEP_REALTIME_. For the Ada Language implementation a set of `Boolean` subtypes contained in package `POSIX_Options` (defined in POSIX.5c, section 2.5) shall be used to specify the presence or absence of each option in this profile.

6.1.2 Conformance

Conformance to the Minimal Realtime System Profile option shall be indicated as follows:

— For the C-Language implementation the symbol _POSIX_AEP_REALTIME_MINIMAL being defined in the header `<unistd.h>`.

— For the Ada Language implementation the `Boolean` subtype `POSIX_Profiles.Realtime_Minimal` subtype having the range `True..True`.

6.1.3 Options

The presence or absence of optional features shall be indicated as follows:

— For the C-language implementation, if any of the following symbols are defined in the header <unistd.h>:

_POSIX_AEP_REALTIME_LANG_C99

_POSIX_AEP_REALTIME_LANG_Ada95

— For the Ada language implementation, if any of the following Boolean subtypes has the range True..True, then the corresponding option is supported:

```
POSIX_Profiles.Realtime_Lang_C99

POSIX_Profiles.Realtime_Lang_Ada95
```

6.2 Operating System Interface Requirements

6.2.1 POSIX.1 Requirements (C Language Option)

The Minimal Realtime System Profile implementation shall include interfaces as defined in POSIX.1 for the following Units of Functionality (see Table 1-1):

Table 6-1: POSIX.1 Units of Functionality Requirements

Unit of Functionality
POSIX_C_LANG_JUMP
POSIX_C_LANG_SUPPORT
POSIX_C_LIB_EXT
POSIX_DEVICE_IO
POSIX_FILE_LOCKING
POSIX_PRIORITY_RANGES
POSIX_SIGNALS
POSIX_SINGLE_PROCESS
POSIX_THREADS_BASE
XSI_THREAD_MUTEX_EXT
XSI_THREADS_EXT

The Minimal Realtime System Profile implementation shall support the following options defined in POSIX.1, by defining the associated symbol with a value greater than zero:

Table 6-2: POSIX.1 Option Requirements

Option
_POSIX_CLOCK_SELECTION
_POSIX_FSYNC
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MONOTONIC_CLOCK
_POSIX_NO_TRUNC
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TIMERS

The value of _POSIX_TIMER_MAX shall be at least 64.

The value of _POSIX_RTSIG_MAX shall be at least 16.

The range of priorities associated with the SCHED_RR scheduling policy shall have at least 31 distinct values that are less than the maximum priority of the SCHED_FIFO policy.

An implementation conforming to PSE51 shall provide a mechanism to configure the system so that the scheduling allocation domain has size one, and so that the binding of threads to scheduling allocation domains remains static. The mechanism by which this requirement is achieved shall be implementation defined. In addition, a PSE51 implementation may provide other configurations or facilities to change the size of the allocation domain and the bindings of threads to allocation domains. For a description of the scheduling allocation domain see the System Interfaces volume of POSIX.1, Section 2.9.2, “Thread Scheduling”.

6.2.2 POSIX.5c Requirements (Ada Language Option)

The Minimal Realtime System Profile implementation shall include interfaces as defined in POSIX.5c for the following units of functionality (see Table 1-2 through Table 1-18):

Table 6-3: POSIX.5c Units of Functionality Requirements

Unit of Functionality
POSIX_ADA_LANG_SUPPORT
POSIX_DEVICE_IO
POSIX_FILE_LOCKING
POSIX_SIGNALS
POSIX_SINGLE_PROCESS

The Minimal Realtime System Profile implementation shall support the following options defined in POSIX.5c, by defining the associated option subtypes to have the range `True..True`, with the exception of the Filename Truncation option for which the associated subtype shall have the range `False..False`:

Table 6-4: POSIX.5c Option Requirements

Option
File Synchronization
Memory Locking
Memory Range Locking
Filename Truncation
Realtime Signals
Semaphores
Shared Memory Objects
Synchronized I/O
Mutexes Support
Mutex Priority Inheritance
Mutex Priority Ceiling
Timers

`POSIX_Limits.Timers_Maxima'First` shall be at least 64.

`POSIX_Limits.Realtime_Signals_Maxima'First` shall be at least 16.

Regarding task priority scheduling, the implementation shall support the following requirements from POSIX.5c and the Ada95 RM:

— The implementation shall support the priority model defined in the Ada95 RM, clause D.1, and the pragmas and package interfaces defined in the Ada95 RM, clauses D.2-D.5.

— The implementation shall meet the requirements of POSIX.5c, section 13.3.1.

The blocking behavior of all reentrant operations defined by POSIX.5c shall be per task, i.e., a blocked task cannot prevent any other task from executing. Therefore, the corresponding `Blocking_Behavior` constants shall have the value `Tasks`. (See POSIX.5c, clause 2.4.1.5.)

Implementations of the PSE51 profile shall support the `POSIX_Profiles` package defined in Annex A of this standard.

The subprogram `POSIX_Signals.Set_Stopped_Child_Signal` shall fail silently.

The subprogram `POSIX_Signals.Stopped_Child_Signal_Enabled` shall return False.

`POSIX_Limits.Groups_Maxima'First` shall be zero.

Subprograms not supported by a given profile shall raise `POSIX_Error`, returning an error code of `Operation_Not_Supported`, except as noted otherwise.

All `Image` and `Value` functions that appear in the packages supported by a profile must be implemented.

Where an overloaded subprogram is required by a unit of functionality, all forms of the subprogram appearing in the referenced clause must be supported, except as otherwise noted.

6.3 Application Constraints

The Minimal Realtime System profile defined in this standard requires only specific Units of Functionality of the required standards. The absence of particular elements of these standards introduces constraints on the use of some of the features of particular operations. This clause defines the constraints that an application strictly conforming to one of the profiles shall observe when using each of the operations required by that profile.

6.3.1 Constraints related to POSIX.1 Interfaces (C Language Option)

The following table defines a set of functions that shall be either reentrant or non-interruptible by signals and shall be async-signal-safe. Therefore applications may invoke them, without restriction, from signal-catching functions. No other function, including those defined in the System Interfaces Volume of POSIX.1, Section 2.4.3, “Signal Actions”, is required to be async-safe in an implementation of the

PSE51 profile, and thus PSE51 Strictly Conforming Applications shall not use them from inside signal handlers.

Table 6-5: Functions required to be async-signal-safe

alarm()	*sigaddset*()	*sigpending*()	*timer_getoverrun*()
clock_gettime()	*sigdelset*()	*sigprocmask*()	*timer_gettime*()
kill()	*sigemptyset*()	*sigqueue*()	*timer_settime*()
raise()	*sigfillset*()	*sigset*()	*times*()
sem_post()	*sigismember*()	*sysconf*()	*uname*()
sigaction()	*signal*()	*time*()	

The *sysconf*() function has the following constraints:

(1) An application strictly conforming to the PSE51 profile shall not call the *sysconf*() function with the parameter _POSIX_VERSION since a meaningful value cannot be returned.[1]

(2) A conforming application must act as if CHILD_MAX=0.

An application strictly conforming to PSE51 shall be considered erroneous if any signal results in abnormal termination of the process because this profiles does not support multiple processes.

An application strictly conforming to PSE51 shall not call the *kill*() function with a negative but not -1 argument because this profile does not require process group functionality.

An application strictly conforming to PSE51 shall be guaranteed that the file mode creation mask for any object created by any process is `S-IRWXU`; that is, the object shall be fully accessible to the creator.

An application strictly conforming to PSE51 shall not use the *open*(), *fopen*(), or *freopen*() functions to create new files, since this profile does not require general file system capabilities.

An application strictly conforming to PSE51 shall use the path or file argument for any function using a file pathname (e.g., *open*()) only to specify the name of the object without any file system semantics implied, since this profile does not require general file system semantics.

An application strictly conforming to PSE51 shall not require that any input/output function (e.g., *fclose*(), *fflush*(), *fgetc*(), *fgets*(), *fopen*(), *fprintf*(), *fputc*(), *fputs*(), *fread*(), *fscanf*(), *fwrite*(), *getc*(), *getchar*(), *gets*(), *open*(), *perror*(), *printf*(), *putc*(), *putchar*(), *puts*(), *read*(), *scanf*(), *vfprintf*(), *vfscanf*(), *vprintf*(), *vscanf*(), *write*()) update an access, creation, or modification time for the device read or written, because this profile requires no interfaces that could query such an access time.

1. Conformance to this profile can be checked with the symbols defined in 6.1.3.

6.3.2 Constraints related to POSIX.5c Interfaces (Ada Language Option)

An application strictly conforming to PSE51 shall not call the functions `POSIX_Configurable_System_Limits.System_POSIX_Version` or `POSIX_Configurable_System_Limits.System_POSIX_Ada_Version`, since a meaningful value cannot be returned.[1]

A conforming application must act as if `POSIX_Limits.Child_Processes_Maxima'Last=0`.

An application strictly conforming to PSE51 shall be considered erroneous if any signal results in abnormal termination of the process because this profile does not support multiple processes.

An application strictly conforming to PSE51 shall not call the form of `POSIX_Signals.Send_Signal` that takes a process group ID as an argument because this profile does not require process group functionality.

An application strictly conforming to PSE51 shall not attempt to bind a signal to a task entry.

An application strictly conforming to PSE51 shall not use the `POSIX_IO.Open_Or_Create` function to create new files, since this profile does not require general file system capabilities.

An application strictly conforming to PSE51 shall use a parameter representing a pathname (such as the `Name` parameter of `POSIX_IO.Open` or `POSIX_IO.Open_Or_Create`) only to specify the name of the object without any file system semantics implied, since this profile does not require general file system semantics.

An application strictly conforming to PSE51 shall not require that any input/output function such as `POSIX_IO.Read`, `POSIX_IO.Generic_Read`, `POSIX_IO.Write`, or `POSIX_IO.Generic_Write`, update an access, creation, or modification time for the device read or written, because this profile requires no interfaces that could query such an access time.

Implementations of PSE51 need not support the `Owner`, `Group`, and `Other` fields of the `form` parameter (See POSIX.5c, clause 8.1.1.2), but may instead raise `Use_Error`. The default value used shall be `Read_Write_Execute`.

Implementations of PSE51 need not support the `File_Structure` field of the `form` parameter (See POSIX.5c, clause 8.1.1.2), but may instead raise `Use_Error`. All files shall default to regular files.

In addition, the following constraints apply to the usage of the predefined Ada I/O packages:

1. Conformance to this profile can be checked with the subtypes defined in 6.1.3.

(1) An application strictly conforming to PSE51 shall not require any of the Input/Output operations (`Read`, `Write`, `Get`, `Put`, etc.) contained in the predefined Ada I/O packages or their instantiations to update an access, creation, or modification time for the device read or written, because this profile requires no interfaces that could query such an access time.

(2) An application strictly conforming to PSE51 shall use the `Name` of the `Open` operations contained in the predefined Ada I/O packages or their instantiations only to specify the name of the object without any file system semantics implied, since this profile does not require general file system capabilities.

(3) An application strictly conforming to PSE51 shall not call any of the `Create` or `Delete` operations contained in the predefined Ada I/O packages or their instantiations, since this profile does not require general file system capabilities.

6.4 Shell and Utility Requirements

An implementation of the Minimal Realtime System Profile is not required to support any of the services described in the Shell and Utilities Volume of POSIX.1.

6.5 Development Platform Requirements

One or more of the development options in 6.5.1 and 6.5.2 shall be implemented.

6.5.1 C Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this standard profile. This platform shall include the POSIX2_C_BIND, POSIX2_C_DEV, and POSIX2_SW_DEV options from the Shell and Utilities Volume of POSIX.1.

6.5.1.1 Option Indicator

The presence of the C Language Development Option shall be indicated by the symbol _POSIX_AEP_REALTIME_LANG_C99 being defined in the required header <unistd.h>. In addition, the presence of the C Language Development Option may be indicated by the subtype POSIX_Profiles.Realtime_Lang_C99 having the range True..True.

6.5.2 Ada Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this profile including applicable portions of the following:

— The Ada95 RM {1}

— POSIX.5c {6}

— The POSIX2_SW_DEV option from the Shell and Utilities Volume of POSIX.1.

6.5.2.1 Option Indicator

The presence of the Ada Language Development Option shall be indicated by the subtype POSIX_Profiles.Realtime_Lang_Ada95 having the range True..True. In addition, the presence of the Ada Language Development Option may be indicated by the symbol _POSIX_AEP_REALTIME_LANG_Ada95 being defined in the header <unistd.h>.

6.6 Rationale for Operating System Requirements (informative)

(This subclause is not a normative part of IEEE Std P1003.13)

6.6.1 Operating System Interface Requirements

After reviewing several commercially available small realtime kernels, it was concluded that the POSIX.1 threads model (with all options enabled, but without a file system) best reflected current industry practice in certain embedded realtime ar-

eas. Instead of full file system support, basic device I/O (read, write, open, close, control) is considered sufficient for kernels of this size. Systems of this size frequently do not include process isolation hardware or software; therefore, multiple processes (as opposed to threads) may not be supportable.

System options that allow an application to be upwards compatible without modifying application source code have been chosen. For example, although the assumed hardware model implies fixed address space without a Memory Management Unit (MMU), the symbol _POSIX_MEMLOCK is still defined. This increases portability of the application code to higher level systems that do not necessarily have the same restrictions.

6.6.1.1 Process Primitives

Because this profile uses the POSIX.1 Threads model only as the mechanism to achieve concurrency, most POSIX.1 process primitives do not apply. This includes the multi-process, pipes, and signal jump units of functionality, as well as the process spawn option.

The *main*() function is needed to allow application-specific information to be passed from boot code to the single (implicit) process (and its threads).

6.6.1.2 Signals

Signal services are a basic mechanism within POSIX-based systems and are required for error and event handling. Realtime systems typically have several logically concurrent software elements executing. Each such entity must respond to several cyclic and/or acyclic stimuli, often in a time-critical manner. Although purely synchronous models can supply such functionality via the use of additional processes or threads, the current realtime practice for asynchronous notification for events such as timeout, message arrival, and hardware interrupt can generally be expected to offer higher performance and lower latency. Realtime Signals provide the reliable high-performance mechanism to support such notification.

The minimum number of realtime signals that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 8, up to 16. The rationale for this increase is that there are many applications that have more than 8 different kinds of events. Doubling the number of required realtime signals should have a minimum impact on the signal management overhead, while significantly increases the number of event kinds that can be used by a strictly conforming application.

6.6.1.3 Process Environment

The functions from the POSIX.1 Process Environment group are deemed necessary to allow an application to determine and configure its system environment. This allows a single version of an application to be run on similar but differing platforms; however, conforming applications must act as if CHILD_MAX=0.

6.6.1.4 Files and Directories

The *open*() function is needed to do basic device I/O and also to provide device initialization. Although this requires some form of name resolution, a full pathname space is specifically not required. Directories also are not required. Units of functionality or options associated with files, such as POSIX_FD_MGMT, POSIX_FIFO, POSIX_FILE_ATTRIBUTES, POSIX_FILE_SYSTEM, POSIX_FILE_SYSTEM_EXT, _POSIX_ADVISORY_INFO, and _POSIX_MAPPED_FILES, are not required.

Since a file system is not a part of this realtime profile, the _POSIX_NO_TRUNC option is applied to the names of devices and shared memory objects.

The File Locking option is required in the C-language option to maintain a consistent and safe way of accessing stdio (*FILE* *) objects from threads, across the four realtime profiles.

6.6.1.5 Input and Output Primitives

The functions contained in the Device I/O unit of functionality are required to do basic I/O and device cleanup.

Asynchronous I/O is not required because it can be easily implemented using threads dedicated to I/O.

6.6.1.6 Synchronized Input and Output

The Synchronized (unbuffered) I/O interface (including the File Synchronization option) is typical for basic device I/O and is required for upward portability.

6.6.1.7 Device- and Class-Specific Functions

POSIX.1 Device- or Class-Specific functions are not required, because small embedded systems usually don't require general-purpose terminal interfaces.

6.6.1.8 System Databases, Users and Groups

Implementations are not required to support more than one user and group id since there are not multiple users and groups. No POSIX.1 System Database functions are required.

6.6.1.9 Synchronization

Mutexes and Condition Variables are required as part of the threads model of concurrency.

The Process Shared option is not required because there is only a single process.

Semaphores are required in the PSE51 profile for synchronization between threads to maintain compatibility with past industry practice. However, mutexes and condition variables are preferred in most current applications. It must be noted that POSIX semaphores do not have the mechanisms built in to avoid unbounded priority inversion when using them for mutually exclusive access to shared resources. Mutexes with the appropriate priority inheritance or priority ceiling (also called priority protection) protocols can be used to avoid this unbounded priority inversion.

Barriers are not required because they can easily be implemented using mutexes and condition variables. Although a direct implementation of barriers can have a significant efficiency benefit in some multiprocessor architectures, a mutex-and-condition-variable implementation will not be significantly slower in most architectures, and thus requiring barriers for all implementations is not justified.

Spin locks are not required because, although they are an efficient synchronization mechanism, they cannot be portably used with the current POSIX.1 interfaces in realtime applications. If a realtime scheduling policy such as SCHED_FIFO or SCHED_RR is used, spin locks may cause deadlock on a single processor. On multiprocessors, to avoid deadlock, it would be necessary for threads using a given lock to be allocated to different processors. There are no standard APIs in the current POSIX.1 to allocate threads to specific processors.

Reader/Writer Locks are not required because they are not designed to avoid unbounded priority inversion, and thus very long delays could occur in realtime applications, with a low but nevertheless non-zero probability. It is expected that a future revision of the POSIX.1 standard will add the priority inheritance and/or priority ceiling options to reader/writer locks, which would eliminate the unbounded priority inversion.

6.6.1.10 Priority Scheduling

Thread priority scheduling is required for realtime applications. The Sporadic Server Scheduling option is also required to enhance support of applications with aperiodic timing requirements. The POSIX_PRIORITY_RANGES unit of functionality is required because threads need to obtain the values of the priority ranges associated with realtime scheduling policies to use those policies.

A common requirement of realtime systems is that they be able to run threads with real-time requirements together with threads with no real-time requirements. One common way of doing this is by having the real-time threads run under the SCHED_FIFO scheduling policy, while the non real-time threads run at a lower priority under the round-robin policy (SCHED_RR) to fairly share the available portion of the processor among them. POSIX requires each policy to have a range of priorities of at least 32 distinct values, but does not impose any requirements on how these priority ranges relate to each other. It could happen that most or all of the SCHED_RR priorities were larger than the SCHED_FIFO priorities, thus making it impossible to mix realtime and non-realtime threads as required above. To solve this problem in a portable way, this profile requires that there are at least 31 SCHED_RR priority levels below the maximum priority of SCHED_FIFO. In this way, a strictly conforming application can use the inclusive priority range [*max_FIFO_prio*, *max_FIFO_prio*-30] with SCHED_FIFO for real-time threads (with a total of 31 priority levels), and then use the priority value min(*max_FIFO_prio*-31,*max_RR_prio*) with the SCHED_RR policy, for the non real-time threads, with guarantee that the latter priority value is valid for the round-robin policy.

Support for a scheduling allocation domain of size one and static binding of threads to allocation domains is required in all the realtime profiles to achieve predictable scheduling behavior. The allocation domain of a thread is the set of processors on which that thread can be scheduled at any given time. The POSIX.1 standard specifies that the scheduling rules have predictable effects only if the allocation domain is of size one; hence the need for this requirement. For single-processor systems the allocation domain is generally of size one and thus the application can meet the requirement just by specifying in the conformance document that the scheduling allocation domain is of size one and that static binding of threads to allocation domains is the default behavior.

6.6.1.11 Process Memory Locking

Process memory locking is inherent in systems following this profile because most PSE51 targets have no MMU and thus swapping is not supported; code and data stay in physical memory until explicitly removed. Nevertheless, memory locking APIs are required for upward portability to allow an application developer to take code intended for a bare PSE51 target and unit test that code on a much larger and more capable platform, perhaps a PSE54, with minimal modification. In those tar-

gets not using a MMU for virtual memory, the locking functions do nothing and always report success, while in the larger profiles there really is memory to be locked. In summary, by requiring this service in the PSE51 profile, it is possible to write portable application code that runs correctly in all the profiles.

6.6.1.12 Shared Memory

Memory Mapped I/O may be implemented using the Shared Memory facility. An implementation is required to provide facilities for creating (shared) memory objects that represent ranges of physical memory that contain device control and status registers or buffers. These facilities encourage the development of portable applications.

Typed Memory objects are not required because they are useful only to systems with special hardware architectures that have various often specialized kinds of memory. Implementors providing support for such special architectures always have the option to provide typed memory objects as an extension.

6.6.1.13 Clocks and Timers

High-resolution timer functions are required in most realtime systems for implementing time management operations such as periodic activations, short duration time-outs, etc. The normal POSIX.1 time management functions *sleep*() and *alarm*() only provide a time resolution of one second, but many realtime systems require finer resolution for specifying time.

The Monotonic Clock is required for realtime applications to ensure that deadlines and timing requirements are not affected by clock jumps.

The Clock Selection option is required to enable choosing the clock on which sleep operations are performed, and to have access to an absolute sleep operation, which is a common requirement in realtime applications with periodic timing requirements.

CPU-Time clocks and timers are required as a means to detect and handle situations in which a thread overruns its assigned maximum execution time. Bounding the execution times of the different threads in the application increases predictability and reliability.

The Timeouts option is a general requirement for realtime applications and thus is required in this profile.

The minimum number of timers that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 32, up to 64, which is the required minimum number of threads. The reason for this increase

is that there are many applications that require one timer per thread (either realtime or CPU-time based).

6.6.1.14 Message Passing

In the PSE51 profile of IEEE Std 1003.13-1998, message queues were required because commercial realtime kernels available at that time with similar functionality to the Minimal Realtime System Profile typically included some form of message queueing mechanism for communication between threads.

However, many embedded realtime applications for small systems do not require message queues and this feature makes the implementation larger. Because message queues can be easily implemented by the application using mutexes and condition variables, this version of the standard has dropped the requirement to support message queues.

6.6.1.15 Threads

The basic assumption in this profile is that the system will consist of a single (implicit) process, with multiple threads. Therefore, all basic thread services are required, except for those related to multiple processes. The POSIX_THREADS_BASE unit of functionality was specified in this document instead of the _POSIX_THREADS option, because this option requires reader/writer locks, but this profile does not.

6.6.1.16 Tracing

Tracing is not required for the PSE51 environment to keep the implementation of this profile small.

6.6.1.17 Networking

Although some small embedded systems require networking services, most don't, so to keep the implementation small, this unit of functionality is not required.

6.6.1.18 Event Management

The *select*() function is usually associated with networking facilities, which are not required for PSE51. Although the function could be used for regular device I/O operations, most kernels that do not have networking services do not support *select*(). Therefore, to keep the implementation small, the event management unit of functionality is not required.

6.6.1.19 Interfaces Related to the Shell and Utilities

Interfaces defined in the POSIX_REGEXP and POSIX_SHELL_FUNC units of functionality are related to shells and utilities, which are not required in this profile; therefore, these units of functionality are not required either.

6.6.1.20 X/Open Units of Functionality and Options

Some XSI Units of Functionality (XSI_C_LANG_SUPPORT, XSI_DEVICE_IO, XSI_DEVICE_SPECIFIC, XSI_FD_MGMT, XSI_FILE_SYSTEM, XSI_IPC, XSI_JOB_CONTROL, XSI_JUMP, XSI_MATH, XSI_MULTI_PROCESS, XSI_SIGNALS, XSI_SINGLE_PROCESS, XSI_SYSTEM_DATABASE, XSI_TIMERS, XSI_USER_GROUPS, XSI_WIDE_CHAR) have interfaces that represent extensions or alternatives to interfaces in other Units of Functionality or POSIX.1 options, and therefore are not necessary for PSE51 environments.

The XSI_DBM unit of functionality includes interfaces for database management that are not required in the PSE51 application environment.

The XSI_DYNAMIC_LINKING unit of functionality is not required for small embedded systems, which usually operate in a static context.

The XSI_I18N unit of functionality provides facilities for natural language messages to the user, which are not required in small embedded systems, which typically do not have general-purpose human interfaces.

The XSI_SYSTEM_LOGGING unit of functionality provides facilities for logging system activities, which are not required in PSE51 environments.

The XSI_THREAD_MUTEX_EXT unit of functionality is required because it has options for controlling the behavior of mutexes under erroneous application use. This capability is interesting for any realtime application, including those targeted at small embedded systems.

The XSI_THREADS_EXT unit of functionality is required because it provides functions to better control a thread's stack. This is considered useful for any realtime application.

The _XOPEN_CRYPT option provides cryptography facilities that are not required in PSE51 environments.

The _XOPEN_LEGACY option provides facilities for backwards compatibility that are not required in PSE51 environments.

The _XOPEN_STREAMS option provides facilities that are mainly related to networking, and thus are not required for PSE51 environments, as discussed above.

6.6.1.21 Language-Specific Services for the C Programming Language

Support for the C Language is required in the C Language option, with the exceptions of the POSIX_C_LANG_MATH and POSIX_C_LANG_WIDE_CHAR units of functionality. The reasons for these exceptions are that these are very large libraries that are not necessary for many of the PSE51 applications.

6.6.1.22 Language-Specific Services for the Ada Programming Language

Support for the Ada language-specific services defined in POSIX.5c is required in the Ada Language option.

6.6.2 Shell and Utility Requirements

Because the Minimal Realtime System Profile is intended for small embedded systems which usually have no terminal or graphical user interface, such a platform would be incapable of executing a shell. In such an environment the utilities described in the Shell and Utilities Volume of POSIX.1 are not usually required.

6.6.3 Development Platform Requirements

The embedded nature of the PSE51 execution platform makes it difficult to use as a development platform. Therefore, the implementation is required to define a development environment in which a PSE51 application can be prepared for execution on the target platform. The development platform depends on the language option chosen by the implementation.

Section 7: Realtime Controller System Profile (PSE52)

7.1 Introduction

This section specifies those standards required for conformance to the Realtime Controller System Profile option and, where applicable, the state of any options contained in those standards.

When a referenced standard specifies services beyond those required by the Realtime Controller System Profile, only those services included in the specified Units of Functionality referenced by this profile shall be required (See Table 1-1 through Table 1-18). All the applicable definitions in POSIX.1 and/or POSIX.5c still apply.

7.1.1 Identification

For the C Language implementation, symbolic names shall be used to specify the presence or absence of each option in this profile. Names reserved for use in this profile begin with the string _POSIX_AEP_REALTIME_. For the Ada language implementation a set of `Boolean` subtypes contained in package `POSIX_Options` (defined in POSIX.5c, clause 2.5) shall be used to specify the presence or absence of each option in this profile.

7.1.2 Conformance

Conformance to the Realtime Controller System Profile option shall be indicated as follows:

— For the C language implementation the symbol _POSIX_AEP_REALTIME_CONTROLLER being defined in the header `<unistd.h>`.

— For the Ada language implementation the `Boolean` subtype `POSIX_Profiles.Realtime_Controller` having the range `True..True`.

7.1.3 Options

The presence or absence of optional features shall be indicated as follows:

— For the C language implementation, if any of the following symbols are defined in the header <unistd.h>, then the corresponding option is supported:

_POSIX_AEP_REALTIME_LANG_C99

_POSIX_AEP_REALTIME_LANG_Ada95

— For the Ada language implementation, if any of the following Boolean subtypes has the range True..True, then the corresponding option is supported:

```
POSIX_Profiles.Realtime_Lang_C99

POSIX_Profiles.Realtime_Lang_Ada95
```

7.2 Operating System Interface Requirements

7.2.1 POSIX.1 Requirements (C language Option)

The Realtime Controller System Profile implementation shall include interfaces as defined in POSIX.1 for the following Units of Functionality (see Table 1-1):

Table 7-1: POSIX.1 Units of Functionality Requirements

Unit of Functionality
POSIX_C_LANG_JUMP
POSIX_C_LANG_MATH
POSIX_C_LANG_SUPPORT
POSIX_DEVICE_IO
POSIX_FD_MGMT
POSIX_FILE_LOCKING
POSIX_FILE_SYSTEM
POSIX_PRIORITY_RANGES
POSIX_SIGNALS
POSIX_SINGLE_PROCESS
POSIX_THREADS_BASE
XSI_THREAD_MUTEX_EXT
XSI_THREADS_EXT

The Realtime Controller System Profile implementation shall support the following options defined in POSIX.1, by defining the associated symbol with a value greater than zero:

Table 7-2: POSIX.1 Option Requirements

Option
_POSIX_CLOCK_SELECTION
_POSIX_FSYNC
_POSIX_MAPPED_FILES
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK
_POSIX_NO_TRUNC
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TIMERS
_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_LOG

The value of _POSIX_TIMER_MAX shall be at least 64.

The value of _POSIX_RTSIG_MAX shall be at least 16.

The range of priorities associated with the SCHED_RR scheduling policy shall have at least 31 distinct values that are less than the maximum priority of the SCHED_FIFO policy.

An implementation conforming to PSE52 shall provide a mechanism to configure the system so that the scheduling allocation domain has size one, and so that the binding of threads to scheduling allocation domains remains static. The mechanism by which this requirement is achieved shall be implementation defined. In addition, a PSE52 implementation may provide other configurations or facilities to change the size of the allocation domain and the bindings of threads to allocation domains. For a description of the scheduling allocation domain see the System Interfaces volume of POSIX.1, Section 2.9.2, “Thread Scheduling”.

7.2.2 POSIX.5c Requirements (Ada Language Option)

The Realtime Controller System Profile implementation shall include interfaces as defined in POSIX.5c for the following Units of Functionality (see Table 1-1):

Table 7-3: POSIX.5c Units of Functionality Requirements

Unit of Functionality
POSIX_ADA_LANG_SUPPORT
POSIX_DEVICE_IO
POSIX_FD_MGMT
POSIX_FILE_SYSTEM
POSIX_SIGNALS
POSIX_SINGLE_PROCESS

The Realtime Controller System Profile implementation shall support the following options defined in POSIX.5c, by defining the associated option subtypes to have the range `True..True`, with the exception of the Filename Truncation option for which the associated subtype shall have the range `False..False`:

Table 7-4: POSIX.5c Option Requirements

Option
File Synchronization
Memory Mapped Files
Memory Locking
Memory Range Locking
Message Queues
Filename Truncation
Realtime Signals
Semaphores
Shared Memory Objects
Synchronized I/O
Mutexes Support
Mutex Priority Inheritance
Mutex Priority Ceiling
Timers

`POSIX_Limits.Timers_Maxima'First` shall be at least 64.

`POSIX_Limits.Realtime_Signals_Maxima'First` shall be at least 16.

Regarding task priority scheduling, the implementation shall support the following requirements from POSIX.5c and the Ada95 RM:

— The implementation shall support the priority model defined in the Ada95 RM, clause D.1, and the pragmas and package interfaces defined in the Ada95 RM, clauses D.2-D.5.

— The implementation shall meet the requirements of POSIX.5c, section 13.3.1.

The blocking behavior of all reentrant operations defined by POSIX.5c shall be per task, i.e., a blocked task cannot prevent any other task from executing. Therefore, the corresponding `Blocking_Behavior` constants shall have the value `Tasks`. (See POSIX.5c, clause 2.4.1.5.)

Implementations of the PSE52 profile shall support the `POSIX_Profiles` package defined in Annex A of this standard.

The subprogram `POSIX_Signals.Set_Stopped_Child_Signal` shall fail silently.

The subprogram `POSIX_Signals.Stopped_Child_Signal_Enabled` shall return False.

`POSIX_Limits.Groups_Maxima'First` shall be zero.

Subprograms not supported by a given profile shall raise `POSIX_Error`, returning an error code of `Operation_Not_Supported`, except as noted otherwise.

All `Image` and `Value` functions that appear in the packages supported by a profile must be implemented.

Where an overloaded subprogram is required by a unit of functionality, all forms of the subprogram appearing in the referenced clause must be supported, except as otherwise noted.

7.3 Application Constraints

The Realtime Controller System profile defined in this standard requires only specific Units of Functionality of the required standards. The absence of particular elements of these standards introduces constraints on the use of some of the features of particular operations. This clause defines the constraints that an application strictly conforming to one of the profiles shall observe when using each of the operations required by that profile.

7.3.1 Constraints related to POSIX.1 Interfaces (C Language Option)

The following table defines a set of functions that shall be either reentrant or non-interruptible by signals and shall be async-signal-safe. Therefore applications may invoke them, without restriction, from signal-catching functions. No other function, including those defined in the System Interfaces Volume of POSIX.1, Section 2.4.3, “Signal Actions”, is required to be async-safe in an implementation of the

PSE52 profile, and thus PSE52 Strictly Conforming Applications shall not use them from inside signal handlers.

Table 7-5: Functions required to be async-signal-safe

alarm()	*sigaddset*()	*sigpending*()	*timer_getoverrun*()
clock_gettime()	*sigdelset*()	*sigprocmask*()	*timer_gettime*()
kill()	*sigemptyset*()	*sigqueue*()	*timer_settime*()
raise()	*sigfillset*()	*sigset*()	*times*()
sem_post()	*sigismember*()	*sysconf*()	*uname*()
sigaction()	*signal*()	*time*()	

The *sysconf*() function has the following constraints:

(1) An application strictly conforming to the PSE52 profile shall not call the *sysconf*() function with the parameter _POSIX_VERSION since a meaningful value cannot be returned.[1]

(2) A conforming application must act as if CHILD_MAX=0.

An application strictly conforming to PSE52 shall be considered erroneous if any signal results in abnormal termination of the process because this profiles does not support multiple processes.

An application strictly conforming to PSE52 shall not call the *kill*() function with a negative but not -1 argument because this profile does not require process group functionality.

An application strictly conforming to PSE52 shall be guaranteed that the file mode creation mask for any object created by any process is `S-IRWXU`; that is, the object shall be fully accessible to the creator.

7.3.2 Constraints related to POSIX.5c Interfaces (Ada Language Option)

An application strictly conforming to PSE52 shall not call the functions `POSIX_Configurable_System_Limits.System_POSIX_Version` or `POSIX_Configurable_System_Limits.System_POSIX_Ada_Version`, since a meaningful value cannot be returned.[2]

A conforming application must act as if `POSIX_Limits.Child_Processes_Maxima'Last=0`.

An application strictly conforming to PSE52 shall be considered erroneous if any signal results in abnormal termination of the process because this profile does not support multiple processes.

---

1. Conformance to this profile can be checked with the symbols defined in 7.1.3.
2. Conformance to this profile can be checked with the subtypes defined in 7.1.3.

An application strictly conforming to PSE52 shall not call the form of `POSIX_Signals.Send_Signal` that takes a process group ID as an argument because this profile does not require process group functionality.

An application strictly conforming to PSE52 shall not attempt to bind a signal to a task entry.

Implementations of PSE52 need not support the `File_Structure` field of the `form` parameter (See POSIX.5c, clause 8.1.1.2), but may instead raise `Use_Error`. All files shall default to regular files.

7.4 Shell and Utility Requirements

An implementation of the Realtime Controller System Profile is not required to support any of the services described in the Shell and Utilities Volume of POSIX.1.

7.5 Development Platform Requirements

One or more of the development options in 7.5.1 and 7.5.2 shall be implemented.

7.5.1 C Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this standard profile. This platform shall include the POSIX2_C_BIND, POSIX2_C_DEV, and POSIX2_SW_DEV options from the Shell and Utilities Volume of POSIX.1.

7.5.1.1 Option Indicator

The presence of the C Language Development Option shall be indicated by the symbol _POSIX_AEP_REALTIME_LANG_C99 being defined in the required header `<unistd.h>`. In addition, the presence of the C Language Development Option may be indicated by the subtype `POSIX_Profiles.Realtime_Lang_C99` having the range `True..True`.

7.5.2 Ada Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this profile including applicable portions of the following:

— The Ada95 RM {1}

— POSIX.5c {6}

— The POSIX2_SW_DEV option from the Shell and Utilities Volume of POSIX.1.

7.5.2.1 Option Indicator

The presence of the Ada Language Development Option shall be indicated by the subtype `POSIX_Profiles.Realtime_Lang_Ada95` having the range `True..True`. In addition, the presence of the Ada Language Development Option may be indicated by the symbol _POSIX_AEP_REALTIME_LANG_Ada95 being defined in the header `<unistd.h>`.

7.6 Rationale for Operating System Requirements (informative)

(This subclause is not a normative part of IEEE Std P1003.13)

7.6.1 Operating System Interface Requirements

This model introduces system functionality that is more sophisticated than in the Minimal Realtime System Profile, specifically in the area of I/O. Two general categories of services are added.

The first extension is support for a simplified file and directory system. These features are used in applications that require an alterable file name space, typically in systems that support secondary storage and require the ability to create, change, and delete named regular files located on a storage device. The included functions allow the creation, deletion, and changing of file attributes of regular files.

This profile assumes the following hardware model: one or more processors with local memory and one or more serial interfaces. (It is anticipated that the serial interface(s) may be removed in final production systems.) Driver-level I/O to stan-

dard and non-standard devices are supported. In addition, a file system device is supported. The hardware is not required to provide memory management.

7.6.1.1 Process Primitives

Because this profile uses the POSIX.1 Threads model only as the mechanism to achieve concurrency, most POSIX.1 process primitives do not apply. This includes the multi-process, pipes, and signal jump units of functionality, as well as the process spawn option. Although PSE52 has only a single (implicit) process, some interprocess APIs are required to support communication between applications.

The *main*() function is needed to allow application-specific information to be passed from boot code to the single process (and its threads).

7.6.1.2 Signals

Signal services are a basic mechanism within POSIX-based systems and are required for error and event handling. Realtime systems typically have several logically concurrent software elements executing. Each such entity must respond to several cyclic and/or acyclic stimuli, often in a time-critical manner. Although purely synchronous models can supply such functionality via the use of additional processes or threads, the current realtime practice for asynchronous notification for events such as timeout, message arrival, and hardware interrupt can generally be expected to offer higher performance and lower latency. Realtime Signals provide the reliable high-performance mechanism to support such notification.

The minimum number of realtime signals that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 8, up to 16. The rationale for this increase is that there are many applications that have more than 8 different kinds of events. Doubling the number of required realtime signals should have a minimum impact on the signal management overhead, while significantly increases the number of event kinds that can be used by a strictly conforming application.

7.6.1.3 Process Environment

The functions from the POSIX.1 Process Environment group are deemed necessary to allow an application to determine and configure its system environment. This allows a single version of an application to be run on similar but differing platforms; however, conforming applications must act as if CHILD_MAX=0.

7.6.1.4 Files and Directories

Since this profile has a file system, all POSIX.1 functions that manage a basic file systems are required. However, the file system in a PSE52 platform is a simplified version of a full POSIX.1 file system, and for this reason the POSIX_FIFO, POSIX_FILE_ATTRIBUTES, and POSIX_FILE_SYSTEM_EXT, units of functionality and the _POSIX_ADVISORY_INFO option are not required.

The File Locking option is required in the C-language option to maintain a consistent and safe way of accessing stdio (*FILE* *) objects from threads, across the four realtime profiles.

7.6.1.5 Input and Output Primitives

The functions contained in the Device I/O and File Descriptor Management units of functionality are required to do basic I/O and device cleanup.

Asynchronous I/O is not required because it can be easily implemented using threads dedicated to I/O.

7.6.1.6 Synchronized Input and Output

The Synchronized (unbuffered) I/O interface (including the File Synchronization option) is typical for basic device I/O and is required for upward portability.

Those realtime systems that use file management systems will frequently require synchronized I/O to provide data integrity and/or relinquish resources to other users. Synchronized I/O as defined in POSIX.1 provides these mechanisms.

7.6.1.7 Device- and Class-Specific Functions

POSIX.1 Device- or Class-Specific functions are not required, because PSE52 systems usually don't require general-purpose terminal interfaces.

7.6.1.8 System Databases, Users and Groups

Implementations are not required to support more than one user and group id since there are not multiple users and groups. No POSIX.1 System Database functions are required.

7.6.1.9 Synchronization

Mutexes and Condition Variables are required as part of threads model of concurrency.

The Process Shared option is not required because there is only a single process.

Semaphores are required in the PSE52 profile for synchronization between threads to maintain compatibility with past industry practice. However, mutexes and conditional variables are preferred in most current applications. It must be noted that POSIX semaphores do not have the mechanisms built in to avoid unbounded priority inversion when using them for mutually exclusive access to shared resources. Mutexes with the appropriate priority inheritance or priority ceiling (also called priority protection) protocols can be used to avoid this unbounded priority inversion.

Barriers are not required because they can easily be implemented using mutexes and condition variables. Although a direct implementation of barriers can have a significant efficiency benefit in some multiprocessor architectures, a mutex-and-condition-variable implementation will not be significantly slower in most architectures, and thus requiring barriers for all implementations is not justified.

Spin locks are not required because, although they are an efficient synchronization mechanism, they cannot be portably used with the current POSIX.1 interfaces in realtime applications. If a realtime scheduling policy such as SCHED_FIFO or SCHED_RR is used, spin locks may cause deadlock on a single processor. On multiprocessors, to avoid deadlock, it would be necessary for threads using a given lock to be allocated to different processors. There are no standard APIs in the current POSIX.1 to allocate threads to specific processors.

Reader/Writer Locks are not required because they are not designed to avoid unbounded priority inversion, and thus very long delays could occur in realtime applications, with a low but nevertheless non-zero probability. It is expected that a future revision of the POSIX.1 standard will add the priority inheritance and/or priority ceiling options to reader/writer locks, which would eliminate the unbounded priority inversion.

7.6.1.10 Priority Scheduling

Thread priority scheduling is required for realtime applications. The Sporadic Server Scheduling option is also required to enhance support of applications with aperiodic timing requirements. The POSIX_PRIORITY_RANGES unit of functionality is required because threads need to obtain the values of the priority ranges associated with realtime scheduling policies to use those policies.

A common requirement of realtime systems is that they be able to run threads with real-time requirements together with threads with no real-time requirements. One

common way of doing this is by having the real-time threads run under the SCHED_FIFO scheduling policy, while the non real-time threads run at a lower priority under the round-robin policy (SCHED_RR) to fairly share the available portion of the processor among them. POSIX requires each policy to have a range of priorities of at least 32 distinct values, but does not impose any requirements on how these priority ranges relate to each other. It could happen that most or all of the SCHED_RR priorities were larger than the SCHED_FIFO priorities, thus making it impossible to mix realtime and non-realtime threads as required above. To solve this problem in a portable way, this profile requires that there are at least 31 SCHED_RR priority levels below the maximum priority of SCHED_FIFO. In this way, a strictly conforming application can use the inclusive priority range [*max_FIFO_prio*, *max_FIFO_prio*-30] with SCHED_FIFO for real-time threads (with a total of 31 priority levels), and then use the priority value min(*max_FIFO_prio*-31,*max_RR_prio*) with the SCHED_RR policy, for the non real-time threads, with guarantee that the latter priority value is valid for the round-robin policy.

Support for a scheduling allocation domain of size one and static binding of threads to allocation domains is required in all the realtime profiles to achieve predictable scheduling behavior. The allocation domain of a thread is the set of processors on which that thread can be scheduled at any given time. The POSIX.1 standard specifies that the scheduling rules have predictable effects only if the allocation domain is of size one; hence the need for this requirement. For single-processor systems the allocation domain is generally of size one and thus the application can meet the requirement just by specifying in the conformance document that the scheduling allocation domain is of size one and that static binding of threads to allocation domains is the default behavior.

7.6.1.11 Process Memory Locking

Process memory locking is inherent in systems following this profile because most PSE52 targets have no MMU and thus swapping is not supported; code and data stays in physical memory until explicitly removed. Nevertheless, memory locking APIs are required for upward portability to allow an application developer to take code intended for a bare PSE52 target and unit test that code on a much larger and more capable platform, perhaps a PSE54, with minimal modification. In those targets not using an MMU for virtual memory, the locking functions do nothing and always report success, while in the larger profiles there really is memory to be locked. In summary, by requiring this service in the PSE52 profile, it is possible to write portable application code that runs correctly in all the profiles.

7.6.1.12 Shared Memory

Memory Mapped I/O may be implemented using the Shared Memory facility. An implementation is required to provide facilities for creating (shared) memory objects that represent ranges of physical memory that contain device control and status registers or buffers. These facilities encourage the development of portable applications.

The Memory-Mapped Files option is included because the implementation has file-system capabilities, and memory-mapped files are a convenient paradigm for reading and writing information in applications following this profile. In memory-mapped files, I/O is not managed by the programmer because data can be manipulated as memory. The implementation of memory-mapped files does not require a significant amount of additional memory or execution overhead to achieve the additional capability.

System vendors are expected to implement the chosen interface in a manner that meets the needs of the applications. In particular, a rotating media-based implementation is allowed but not required by the interface definition.

Typed Memory objects are not required because they are useful only to systems with special hardware architectures that have various often specialized kinds of memory. Implementors providing support for such special architectures always have the option to provide typed memory objects as an extension.

7.6.1.13 Clocks and Timers

High-resolution timer functions are required in most realtime systems for implementing time management operations such as periodic activations, short duration time-outs, etc. The normal POSIX.1 time management functions *sleep*() and *alarm*() only provide a time resolution of one second, but many realtime systems require finer resolution for specifying time.

The Monotonic Clock is required for realtime applications to ensure that deadlines and timing requirements are not affected by clock jumps.

The Clock Selection option is required to enable choosing the clock on which sleep operations are performed, and to have access to an absolute sleep operation, which is a common requirement in realtime applications with periodic timing requirements.

CPU-Time clocks and timers are required as a means to detect and handle situations in which a thread overruns its assigned maximum execution time. Delimiting the execution times of the different threads in the application provides temporal partitioning in realtime applications, and thus increases predictability and reliability.

The Timeouts option is a general requirement for realtime applications and thus is required in this profile.

The minimum number of timers that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 32, up to 64, which is the required minimum number of threads. The reason for this increase is that there are many applications that require one timer per thread (either real-time or CPU-time based).

7.6.1.14 Message Passing

Currently available commercial realtime kernels with similar functionality to the Realtime Controller System Profile typically include some form of message queueing mechanism for communication between threads. The POSIX.1 Message Passing offers an appropriate level of performance to provide this functionality.

7.6.1.15 Threads

The basic assumption in this profile is that the system will consist of a single (implicit) process, with multiple threads. Therefore, all basic thread services are required, except for those related to multiple processes. The POSIX_THREADS_BASE unit of functionality was specified in this document instead of the _POSIX_THREADS option, because this option requires reader/writer locks, but this profile does not.

7.6.1.16 Tracing

Tracing is required for the PSE52 environment because most of these systems work in an unattended mode for long periods of time, and tracing provides an excellent mechanism to support post-failure analysis, particularly for failures having a low probability of occurrence.

The Trace Event Filtering option is required for the system to be able to filter out those trace events that are not meaningful for the application, thus making better use of system resources by capturing only the interesting events.

The presence of a file system in the PSE52 profile facilitates the recording of the trace events, through the Trace Log option, which is required for this profile.

7.6.1.17 Networking

Although some small controller systems require networking services, most don't, so to keep the implementation small, this unit of functionality is not required.

7.6.1.18 Event Management

The *select*() function is usually associated with networking facilities, which are not required for PSE52. Although the function could be used for regular device I/O operations, most kernels that do not have networking services do not support *select*(). Therefore, to keep the implementation small, the event management unit of functionality is not required.

7.6.1.19 Interfaces Related to the Shell and Utilities

Interfaces defined in the POSIX_REGEXP and POSIX_SHELL_FUNC units of functionality are related to shells and utilities, which are not required in this profile; therefore, these units of functionality are not required either.

7.6.1.20 X/Open Units of Functionality and Options

Some XSI Units of Functionality (XSI_C_LANG_SUPPORT, XSI_DEVICE_IO, XSI_DEVICE_SPECIFIC, XSI_FD_MGMT, XSI_FILE_SYSTEM, XSI_IPC, XSI_JOB_CONTROL, XSI_JUMP, XSI_MATH, XSI_MULTI_PROCESS, XSI_SIGNALS, XSI_SINGLE_PROCESS, XSI_SYSTEM_DATABASE, XSI_TIMERS, XSI_USER_GROUPS, XSI_WIDE_CHAR) have interfaces that represent extensions or alternatives to interfaces in other Units of Functionality or POSIX.1 options, and therefore are not necessary for PSE52 environments.

The XSI_DBM unit of functionality includes interfaces for database management that are not required in the PSE52 application environment.

The XSI_DYNAMIC_LINKING unit of functionality is not required for small embedded systems, which usually operate in a static context.

The XSI_I18N unit of functionality provides facilities for natural language messages to the user, which are not required in realtime controller systems, which typically do not have general-purpose human interfaces.

The XSI_SYSTEM_LOGGING unit of functionality provides facilities for logging system activities, which are not required in PSE52 environments.

The XSI_THREAD_MUTEX_EXT unit of functionality is required because it has options for controlling the behavior of mutexes under erroneous application use. This capability is interesting for any realtime application, including those targeted at control systems.

The XSI_THREADS_EXT unit of functionality is required because it provides functions to better control a thread's stack. This is considered useful for any realtime application.

The _XOPEN_CRYPT option provides cryptography facilities that are not required in PSE52 environments.

The _XOPEN_LEGACY option provides facilities for backwards compatibility that are not required in PSE52 environments.

The _XOPEN_STREAMS option provides facilities that are mainly related to networking, and thus are not required for PSE52 environments, as discussed above.

7.6.1.21 Language-Specific Services for the C Programming Language

Support for the C Language is required in the C Language option, with the exception of the POSIX_C_LANG_WIDE_CHAR unit of functionality. The reason for this exception is that this is a very large library that is not necessary for many of the PSE52 applications.

7.6.1.22 Language-Specific Services for the Ada Programming Language

Support for the Ada language-specific services defined in POSIX.5c is required in the Ada Language option.

7.6.2 Shell and Utility Requirements

Because the Realtime Controller System Profile is intended for control systems which usually have no terminal or graphical user interface, such a platform would be incapable of executing a shell. In such an environment the utilities described in the Shell and Utilities Volume of POSIX.1 are not usually required.

7.6.3 Development Platform Requirements

The special-purpose nature of the PSE52 execution platform makes it difficult to use as a development platform. Therefore, the implementation is required to define a development environment in which a PSE52 application can be prepared for execution on the target platform. The development platform depends on the language option chosen by the implementation.

Section 8: Dedicated Realtime System Profile (PSE53)

8.1 Introduction

This section specifies those standards required for conformance to the Dedicated Realtime System Profile option and, where applicable, the state of any options contained in those standards.

When a referenced standard specifies services beyond those required by the Dedicated Realtime System Profile, only those services included in the specified Units of Functionality referenced by this profile shall be required (See Table 1-1 through Table 1-18). All the applicable definitions in POSIX.1 and/or POSIX.5c still apply.

8.1.1 Identification

For the C-Language implementation, symbolic names shall be used to specify the presence or absence of each option in this profile. Names reserved for use in this profile begin with the string _POSIX_AEP_REALTIME_. For the Ada Language implementation a set of `Boolean` subtypes contained in package `POSIX_Options` (defined in POSIX.5c, section 2.5) shall be used to specify the presence or absence of each option in this profile.

8.1.2 Conformance

Conformance to the Dedicated Realtime System Profile option shall be indicated as follows:

— For the C-Language implementation the symbol _POSIX_AEP_REALTIME_DEDICATED being defined in the header `<unistd.h>`.

— For the Ada Language implementation the `Boolean` subtype `POSIX_Profiles.Realtime_Dedicated` subtype having the range `True..True`.

8.1.3 Options

The presence or absence of optional features shall be indicated as follows:

— For the C-language implementation, if any of the following symbols are defined in the header <unistd.h>:

_POSIX_AEP_REALTIME_LANG_C99

_POSIX_AEP_REALTIME_LANG_Ada95

— For the Ada language implementation, if any of the following Boolean subtypes has the range True..True, then the corresponding option is supported:

```
POSIX_Profiles.Realtime_Lang_C99

POSIX_Profiles.Realtime_Lang_Ada95
```

8.2 Operating System Interface Requirements

8.2.1 POSIX.1 Requirements (C Language Option)

The Dedicated Realtime System Profile implementation shall include interfaces as defined in POSIX.1 for the following Units of Functionality (see Table 1-1)

Table 8-1: POSIX.1 Units of Functionality Requirements

Unit of Functionality
POSIX_C_LANG_JUMP
POSIX_C_LANG_MATH
POSIX_C_LANG_SUPPORT
POSIX_DEVICE_IO
POSIX_EVENT_MGMT
POSIX_FD_MGMT
POSIX_FILE_LOCKING
POSIX_FILE_SYSTEM
POSIX_MULTI_PROCESS
POSIX_NETWORKING
POSIX_PIPE
POSIX_SIGNALS
POSIX_SIGNAL_JUMP
POSIX_SINGLE_PROCESS
POSIX_THREADS_BASE
XSI_THREAD_MUTEX_EXT

Table 8-1: POSIX.1 Units of Functionality Requirements (Continued)

Unit of Functionality
XSI_THREADS_EXT

The Dedicated Realtime System Profile implementation shall support the following options defined in POSIX.1, by defining the associated symbol with a value greater than zero:

Table 8-2: POSIX.1 Option Requirements

Option
_POSIX_ASYNCHRONOUS_IO
_POSIX_CLOCK_SELECTION
_POSIX_CPUTIME
_POSIX_FSYNC
_POSIX_MAPPED_FILES
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK
_POSIX_NO_TRUNC
_POSIX_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING
_POSIX_RAW_SOCKETS
_POSIX_REALTIME_SIGNALS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PROCESS_SHARED
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TIMERS
_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_LOG

The value of _POSIX_TIMER_MAX shall be at least 64.

The value of _POSIX_RTSIG_MAX shall be at least 16.

The range of priorities associated with the SCHED_RR scheduling policy shall have at least 31 distinct values that are less than the maximum priority of the SCHED_FIFO policy.

An implementation conforming to PSE53 shall support the PTHREAD_SCOPE_SYSTEM scheduling contention scope. In addition, it may support PTHREAD_SCOPE_PROCESS. For a description of the scheduling contention scope see the System Interfaces volume of POSIX.1, Section 2.9.2, “Thread Scheduling”.

An implementation conforming to PSE53 shall provide a mechanism to configure the system so that the scheduling allocation domain has size one, and so that the binding of threads to scheduling allocation domains remains static. The mechanism by which this requirement is achieved shall be implementation defined. In addition, a PSE53 implementation may provide other configurations or facilities to change the size of the allocation domain and the bindings of threads to allocation domains. For a description of the scheduling allocation domain see the System Interfaces volume of POSIX.1, Section 2.9.2, “Thread Scheduling”.

8.2.2 POSIX.5c Requirements (Ada Language Option)

The Dedicated Realtime System Profile implementation shall include interfaces as defined in POSIX.5c for the following units of functionality (see Table 1-2 through Table 1-18):

Table 8-3: POSIX.5c Units of Functionality Requirements

Unit of Functionality
POSIX_ADA_LANG_SUPPORT
POSIX_DEVICE_IO
POSIX_EVENT_MGMT
POSIX_FD_MGMT
POSIX_FILE_SYSTEM
POSIX_MULTI_PROCESS[a]
POSIX_NETWORKING
POSIX_PIPE
POSIX_SIGNALS
POSIX_SINGLE_PROCESS

a. The POSIX_MULTI_PROCESS unit of functionality shall be supported, with the provision that the package `POSIX_Unsafe_Process_Primitives` is not required

The Dedicated Realtime System Profile implementation shall support the following options defined in POSIX.5c, by defining the associated option subtypes to have

the range `True..True`, with the exception of the Filename Truncation option for which the associated subtype shall have the range `False..False`:

Table 8-4: POSIX.5c Option Requirements

Option
Asynchronous I/O
File Synchronization
Memory Mapped Files
Memory Locking
Memory Range Locking
Memory Protection
Message Queues
Filename Truncation
Prioritized I/O
Priority Process Scheduling
Realtime Signals
Semaphores
Shared Memory Objects
Synchronized I/O
Mutexes Support
Mutex Priority Inheritance
Mutex Priority Ceiling
Process Shared
Timers

`POSIX_Limits.Timers_Maxima'First` shall be at least 64.

`POSIX_Limits.Realtime_Signals_Maxima'First` shall be at least 16.

Regarding task priority scheduling, the implementation shall support the following requirements from POSIX.5c and the Ada95 RM:

— The implementation shall support the priority model defined in the Ada95 RM, clause D.1, and the pragmas and package interfaces defined in the Ada95 RM, clauses D.2-D.5.

— The implementation shall meet the requirements of POSIX.5c, section 13.3.1.

Implementations of the PSE53 profile shall support the `POSIX_Profiles` package defined in Annex A of this standard.

The subprogram `POSIX_Signals.Set_Stopped_Child_Signal` shall fail silently.

The subprogram `POSIX_Signals.Stopped_Child_Signal_Enabled` shall return False.

`POSIX_Limits.Groups_Maxima'First` shall be zero.

Subprograms not supported by a given profile shall raise `POSIX_Error`, returning an error code of `Operation_Not_Supported`, except as noted otherwise.

All `Image` and `Value` functions that appear in the packages supported by a profile must be implemented.

Where an overloaded subprogram is required by a unit of functionality, all forms of the subprogram appearing in the referenced clause must be supported, except as otherwise noted.

8.3 Application Constraints

The Dedicated Realtime System profile defined in this standard requires only specific units of functionality of the required standards. The absence of particular elements of these standards introduces constraints on the use of some of the features of particular operations. This clause defines the constraints that an application strictly conforming to one of the profiles shall observe when using each of the operations required by that profile.

8.3.1 Constraints related to POSIX.1 Interfaces (C Language Option)

The *sysconf*() function has the following constraints:

(1) An application strictly conforming to the PSE53 profile shall not call the *sysconf*() function with the parameter _POSIX_VERSION since a meaningful value cannot be returned.[1]

An application strictly conforming to PSE53 shall not call the *kill*() function with a negative argument because this profile does not require process group functionality.

An application strictly conforming to PSE53, shall be guaranteed that the file mode creation mask for any object created by any process is `S-IRWXU`; that is, the object shall be fully accessible to the creator.

8.3.2 Constraints related to POSIX.5c Interfaces (Ada Language Option)

An application strictly conforming to PSE53 shall not call the functions `POSIX_Configurable_System_Limits.System_POSIX_Version` or

1. Conformance to this profile can be checked with the symbols defined in 8.1.3.

`POSIX_Configurable_System_Limits.System_POSIX_Ada_Version`, since a meaningful value cannot be returned.[1]

An application strictly conforming to PSE53 shall not call the subprograms contained in the package `Posix_Unsafe_Process_Primitives`, but shall instead rely upon either `Posix_Process_Primitives.Start_Process` or `Posix_Process_Primitives.Start_Process_Search` to create new processes.

An application strictly conforming to PSE53 shall not call the form of `POSIX_Signals.Send_Signal` that takes a process group ID as an argument because this profile does not require process group functionality.

An application strictly conforming to PSE53 shall not attempt to bind a signal to a task entry.

Implementations of PSE53 need not support the `File_Structure` field of the `form` parameter (See POSIX.5c, clause 8.1.1.2), but may instead raise `Use_Error`. All files shall default to regular files.

8.4 Shell and Utility Requirements

An implementation of the Dedicated Realtime System Profile is not required to support any of the services described in the Shell and Utilities Volume of POSIX.1.

8.5 Development Platform Requirements

One or more of the development options in 8.5.1 and 8.5.2 shall be implemented.

8.5.1 C Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this standard profile. This platform shall include the POSIX2_C_BIND, POSIX2_C_DEV, and POSIX2_SW_DEV options from the Shell and Utilities Volume of POSIX.1.

1. Conformance to this profile can be checked with the subtypes defined in 8.1.3.

8.5.1.1 Option Indicator

The presence of the C Language Development Option shall be indicated by the symbol _POSIX_AEP_REALTIME_LANG_C99 being defined in the required header <unistd.h>. In addition, the presence of the C Language Development Option may be indicated by the subtype POSIX_Profiles.Realtime_Lang_C99 having the range True..True.

8.5.2 Ada Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this profile including applicable portions of the following:

— The Ada95 RM {1}

— POSIX.5c {6}

— The POSIX2_SW_DEV option from the Shell and Utilities Volume of POSIX.1.

8.5.2.1 Option Indicator

The presence of the Ada Language Development Option shall be indicated by the subtype POSIX_Profiles.Realtime_Lang_Ada95 having the range True..True. In addition, the presence of the Ada Language Development Option may be indicated by the symbol _POSIX_AEP_REALTIME_LANG_Ada95 being defined in the header <unistd.h>.

8.6 Rationale for Operating System Requirements (informative)

(This subclause is not a normative part of IEEE Std P1003.13)

8.6.1 Operating System Interface Requirements

This profile is based on existing practice in large embedded systems (a single user is assumed). Traditionally, these applications are designed to run with either a home-grown or standard operating system providing process, I/O, time, memory,

and event management services. These applications require support for a simplified file system.

Where convenient, the AEP profile working group has chosen system options that allow an application to be upwardly portable without modifying application source code.

8.6.1.1 Process Primitives

Applications that correspond to the Dedicated Realtime System Environment are usually large embedded systems that require multiple processes for handling multiple, concurrent activities with independent address spaces. The process control functions (which include process creation and execution) are the basic operating system services required to support multiple processes, and are therefore required in these systems.

8.6.1.2 Signals

Signal services are a basic mechanism within POSIX-based systems and are required for error and event handling. Realtime systems typically have several logically concurrent software elements executing. Each such entity must respond to several cyclic and/or acyclic stimuli, often in a time-critical manner. Although purely synchronous models can supply such functionality via the use of additional processes or threads, the current realtime practice for asynchronous notification for events such as timeout, message arrival, and hardware interrupt can generally be expected to offer higher performance and lower latency. Realtime Signals provide the reliable high-performance mechanism to support such notification.

The minimum number of realtime signals that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 8, up to 16. The rationale for this increase is that there are many applications that have more than 8 different kinds of events. Doubling the number of required realtime signals should have a minimum impact on the signal management overhead, while significantly increases the number of event kinds that can be used by a strictly conforming application.

8.6.1.3 Process Environment

The functions from the POSIX.1 Process Environment group are deemed necessary to allow an application to determine and configure its system environment. This allows a single version of an application to be run on similar but differing platforms.

Since these systems require multiple processes, but not users or groups, the functions defined by the POSIX_MULTI_PROCESS unit of functionality are required.

8.6.1.4 Files and Directories

Since this profile has a file system, all POSIX.1 functions that manage a basic file systems are required. However, the file system in a PSE53 platform is a simplified version of a full POSIX.1 file system, and for this reason the POSIX_FIFO, POSIX_FILE_ATTRIBUTES, and POSIX_FILE_SYSTEM_EXT, units of functionality and the _POSIX_ADVISORY_INFO option are not required.

The File Locking option is required in the C-language option to maintain a consistent and safe way of accessing stdio (*FILE* *) objects from threads, across the four realtime profiles.

The File Descriptor Management unit of functionality is included to aid the handling of file descriptors across the process creation and program execution operations.

8.6.1.5 Input and Output Primitives

The functions contained in the Device I/O unit of functionality are required to do basic I/O and device cleanup.

Although asynchronous I/O can be easily implemented using threads dedicated to I/O, it is required in the PSE53 profile to support portability of applications that may have been developed before POSIX threads implementations were widely available.

8.6.1.6 Synchronized Input and Output

The Synchronized (unbuffered) I/O interface (including the File Synchronization option) is typical for basic device I/O and is required for upward portability.

Those realtime systems that use file management systems will frequently require synchronized I/O to provide data integrity and/or relinquish resources to other users. Synchronized I/O as defined in POSIX.1 provides these mechanisms.

8.6.1.7 Device- and Class-Specific Functions

POSIX.1 Device- or Class-Specific functions are not required, because embedded systems usually don't require general-purpose terminal interfaces.

8.6.1.8 System Databases, Users and Groups

Implementations are not required to support more than one user and group id since there are not multiple users and groups. No POSIX.1 System Database functions are required.

8.6.1.9 Synchronization

Mutexes and Condition Variables are required as part of threads model of concurrency.

Semaphores are required to support portability of applications that might be using this mechanism instead of the preferred mutexes and condition variables. It must be noted, however, that POSIX semaphores do not have the mechanisms built in to avoid unbounded priority inversion when using them for mutually exclusive access to shared resources. Mutexes with the appropriate priority inheritance or priority ceiling (also called priority protection) protocols can be used to avoid this unbounded priority inversion. The Process Shared option is required to support applications requiring this mechanism for synchronization across different processes.

Barriers are not required because they can easily be implemented using mutexes and condition variables. Although a direct implementation of barriers can have a significant efficiency benefit in some multiprocessor architectures, a mutex-and-condition-variable implementation will not be significantly slower in most architectures, and thus requiring barriers for all implementations is not justified.

Spin locks are not required because, although they are an efficient synchronization mechanism, they cannot be portably used with the current POSIX.1 interfaces in realtime applications. If a realtime scheduling policy such as SCHED_FIFO or SCHED_RR is used, spin locks may cause deadlock on a single processor. On multiprocessors, to avoid deadlock, it would be necessary for threads using a given lock to be allocated to different processors. There are no standard APIs in the current POSIX.1 to allocate threads to specific processors.

Reader/Writer Locks are not required because they are not designed to avoid unbounded priority inversion, and thus very long delays could occur in realtime applications, with a low but nevertheless non-zero probability. It is expected that a future revision of the POSIX.1 standard will add the priority inheritance and/or

priority ceiling options to reader/writer locks, which would eliminate the unbounded priority inversion.

8.6.1.10 Priority Scheduling

Thread and process priority scheduling are required for realtime applications. The Sporadic Server Scheduling option is also required for processes and threads, to enhance support of applications with aperiodic timing requirements. The POSIX_PRIORITY_RANGES unit of functionality is not required because its functions are already included in the required _POSIX_PRIORITY_SCHEDULING option.

A common requirement of realtime systems is that they be able to run threads or processes with real-time requirements together with threads with no real-time requirements. One common way of doing this is by having the real-time threads run under the SCHED_FIFO scheduling policy, while the non real-time threads run at a lower priority under the round-robin policy (SCHED_RR) to fairly share the available portion of the processor among them. POSIX requires each policy to have a range of priorities of at least 32 distinct values, but does not impose any requirements on how these priority ranges relate to each other. It could happen that most or all of the SCHED_RR priorities were larger than the SCHED_FIFO priorities, thus making it impossible to mix realtime and non-realtime threads as required above. To solve this problem in a portable way, this profile requires that there are at least 31 SCHED_RR priority levels below the maximum priority of SCHED_FIFO. In this way, a strictly conforming application can use the inclusive priority range [*max_FIFO_prio*, *max_FIFO_prio*-30] with SCHED_FIFO for real-time threads (with a total of 31 priority levels), and then use the priority value min(*max_FIFO_prio*-31,*max_RR_prio*) with the SCHED_RR policy, for the non real-time threads, with guarantee that the latter priority value is valid for the round-robin policy.

The implementation is required to support the PTHREAD_SYSTEM_SCOPE thread-scheduling contention scope. The contention scope of a thread defines the set of threads with which the thread competes for use of the processing resources. A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends for resources with all other threads in the system that have the same scheduling allocation domain. This allows a consistent scheduling of threads across the system and therefore a predictable timing behavior. As a consequence, this is the preferred method for realtime systems.

The current POSIX.1 specification allows implementations to support either system-wide or process-wide contention scope, or both. This represents a compromise that tries to address the requirements of both realtime and non-realtime applications, but introduces a potential source for non portability. Because the realtime profiles are specifically targeted at realtime systems, the system-wide contention scope option is required in the profiles that support multiple processes. Process-

wide contention scope may also be provided, perhaps for the non realtime threads of the application.

Support for a scheduling allocation domain of size one and static binding of threads to allocation domains is required in all the realtime profiles to achieve predictable scheduling behavior. The allocation domain of a thread is the set of processors on which that thread can be scheduled at any given time. The POSIX.1 standard specifies that the scheduling rules have predictable effects only if the allocation domain is of size one; hence the need for this requirement. For single-processor systems the allocation domain is generally of size one and thus the application can meet the requirement just by specifying in the conformance document that the scheduling allocation domain is of size one and that static binding of threads to allocation domains is the default behavior.

8.6.1.11 Process Memory Locking

Realtime processes must be able to guarantee memory residency to reduce the latency for instruction fetches, data access, I/O operations, etc. The mechanism described in the POSIX.1 Process Memory Locking extension will satisfy this requirement.

8.6.1.12 Shared Memory

The Shared Memory Objects option provides the capability for more than one execution entity to share memory, without incurring the overhead of the shared memory object on permanent media. Memory Mapped I/O may be implemented using the Shared Memory facility. An implementation must provide facilities for creating a block of physical memory in which the application may place devices and facilities for binding to a user-provided pathname through which a device may subsequently be opened as a Shared Memory special file, and mapped into the process address space for the purpose of performing I/O or other functions from applications programs.

Typed Memory objects are not required because they are useful only to systems with special hardware architectures that have various often specialized kinds of memory. Implementors providing support for such special architectures always have the option to provide typed memory objects as an extension.

8.6.1.13 Clocks and Timers

High-resolution timer functions are required in most realtime systems for implementing time management operations such as periodic activations, short duration

time-outs, etc. The normal POSIX.1 time management functions *sleep*() and *alarm*() only provide a time resolution of one second, but many realtime systems require finer resolution for specifying time.

The Monotonic Clock is required for realtime applications to ensure that deadlines and timing requirements are not affected by clock jumps.

The Clock Selection option is required to enable choosing the clock on which sleep operations are performed, and to have access to an absolute sleep operation, which is a common requirement in realtime applications with periodic timing requirements.

CPU-Time clocks and timers are required as a means to detect and handle situations in which a thread overruns its assigned maximum execution time. Delimiting the execution times of the different threads in the application provides temporal partitioning in realtime applications, and thus increases predictability and reliability.

The Timeouts option is a general requirement for realtime applications and thus is required in this profile.

The minimum number of per-process timers that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 32, up to 64, which is the required minimum number of threads per process. The reason for this increase is that there are many applications that require one timer per thread (either realtime or CPU-time based).

8.6.1.14 Message Passing

These realtime systems typically include some form of message queuing mechanism for communication among processes or threads. The POSIX.1 message passing offers an appropriate level of performance to provide this functionality.

8.6.1.15 Threads

The basic assumption in this profile is that the system will consist of one or more processes with multiple threads. Therefore, all thread services are required. The POSIX_THREADS_BASE unit of functionality was specified in this document instead of the _POSIX_THREADS option, because this option requires reader/writer locks, but this profile does not.

8.6.1.16 Tracing

Tracing is required for the PSE53 environment because most of these systems work in an unattended mode for long periods of time, and tracing provides an excellent mechanism to support post-failure analysis, particularly for failures having a low probability of occurrence.

The Trace Event Filtering option is required for the system to be able to filter out those trace events that are not meaningful for the application, thus making better use of system resources by capturing only the interesting events.

Because the PSE53 profile does not require general file system capabilities, the Trace Log option is not required for this profile.

8.6.1.17 Networking

Today, most of the platforms and applications belonging to the PSE53 environment require network communications, and thus the networking unit of functionality is required in this profile. The Raw Sockets option is required to aid reconfiguration of networked applications, and to implement special protocols directly, without the weight of a full protocol stack. The Internet Protocol Version 6 option is not required because most applications are not using this version of the protocol yet.

8.6.1.18 Event Management

The *select*() function is usually associated with networking facilities, which are required for PSE53, and thus the Event Management unit of functionality is required in the PSE53 environment.

8.6.1.19 Interfaces Related to the Shell and Utilities

Interfaces defined in the POSIX_REGEXP and POSIX_SHELL_FUNC units of functionality are related to shells and utilities, which are not required in this profile; therefore, these units of functionality are not required either.

8.6.1.20 X/Open Units of Functionality and Options

Some XSI Units of Functionality (XSI_C_LANG_SUPPORT, XSI_DEVICE_IO, XSI_DEVICE_SPECIFIC, XSI_FD_MGMT, XSI_FILE_SYSTEM, XSI_IPC, XSI_JOB_CONTROL, XSI_JUMP, XSI_MATH, XSI_MULTI_PROCESS, XSI_SIGNALS, XSI_SINGLE_PROCESS, XSI_SYSTEM_DATABASE, XSI_TIMERS, XSI_USER_GROUPS, XSI_WIDE_CHAR) have interfaces that represent extensions or alternatives to interfaces in other Units of Functionality or POSIX.1 options, and therefore are not necessary for PSE53 environments.

The XSI_DBM unit of functionality includes interfaces for database management that are not required in the PSE53 application environment.

The XSI_DYNAMIC_LINKING unit of functionality is not required for embedded systems, which usually operate in a static context.

The XSI_I18N unit of functionality provides facilities for natural language messages to the user, which are not required in embedded systems, which typically do not have general-purpose human interfaces.

The XSI_SYSTEM_LOGGING unit of functionality provides facilities for logging system activities, which are not required in PSE53 environments.

The XSI_THREAD_MUTEX_EXT unit of functionality is required because it has options for controlling the behavior of mutexes under erroneous application use. This capability is interesting for any realtime application, including those targeted at small embedded systems.

The XSI_THREADS_EXT unit of functionality is required because it provides functions to better control a thread's stack. This is considered useful for any realtime application.

The _XOPEN_CRYPT option provides cryptography facilities that are not required in most PSE53 environments.

The _XOPEN_LEGACY option provides facilities for backwards compatibility that are not required in PSE53 environments.

The _XOPEN_STREAMS option provides facilities that are not required in most PSE53 environments.

8.6.1.21 Language-Specific Services for the C Programming Language

Support for the C Language is required in the C language option, with the exception of the POSIX_C_LANG_WIDE_CHAR unit of functionality. The reason for this exception is that this is a very large library that is not necessary for many of the PSE53 applications.

8.6.1.22 Language-Specific Services for the Ada Programming Language

Support for the Ada language-specific services defined in POSIX.5c is required in the Ada language option.

8.6.2 Shell and Utility Requirements

Because the Dedicated Realtime System Profile is intended for embedded systems which usually have no terminal or general-purpose graphical user interface, such a platform would be incapable of executing a shell. In such an environment the utilities described in the Shell and Utilities Volume of POSIX.1 are not usually required.

8.6.3 Development Platform Requirements

The embedded nature of the PSE53 execution platform makes it difficult to use as a development platform. Therefore, the implementation is required to define a development environment in which a PSE53 application can be prepared for execution on the target platform. The development platform depends on the language option chosen by the implementation.

Section 9: Multi-Purpose Realtime System Profile (PSE54)

9.1 Introduction

This section specifies those standards required for conformance to the Multi-Purpose Realtime System Profile option and, where applicable, the state of any options contained in those standards.

When a referenced standard specifies services beyond those required by the Multi-Purpose Realtime System Profile, only those services included in the specified Units of Functionality referenced by this profile shall be required (See Table 1-1 through Table 1-18). All the applicable definitions in POSIX.1 and/or POSIX.5c still apply.

9.1.1 Identification

For the C-Language implementation, symbolic names shall be used to specify the presence or absence of each option in this profile. Names reserved for use in this profile begin with the string _POSIX_AEP_REALTIME_. For the Ada Language implementation a set of `Boolean` subtypes contained in package `POSIX_Options` (defined in POSIX.5c, section 2.5) shall be used to specify the presence or absence of each option in this profile.

9.1.2 Conformance

Conformance to the Multi-Purpose Realtime System Profile option shall be indicated as follows:

— For the C-Language implementation the symbol _POSIX_AEP_REALTIME_MULTI being defined in the header <unistd.h>.

— For the Ada Language implementation the `Boolean` subtype `POSIX_Profiles.Realtime_Multi` subtype having the range `True..True`.

9.1.3 Options

The presence or absence of optional features shall be indicated as follows:

— For the C-language implementation, if any of the following symbols are defined in the header <unistd.h>:

_POSIX_AEP_REALTIME_LANG_C99

_POSIX_AEP_REALTIME_LANG_Ada95

— For the Ada language implementation, if any of the following Boolean subtypes has the range True..True, then the corresponding option is supported:

```
POSIX_Profiles.Realtime_Lang_C99

POSIX_Profiles.Realtime_Lang_Ada95
```

9.2 Operating System Interface Requirements

9.2.1 POSIX.1 Requirements (C Language Option)

The Multi-Purpose Realtime System Profile implementation shall include interfaces as defined in POSIX.1 for the following Units of Functionality (see Table 1-1)

Table 9-1: POSIX.1 Units of Functionality Requirements

Unit of Functionality
POSIX_C_LANG_JUMP
POSIX_C_LANG_MATH
POSIX_C_LANG_SUPPORT
POSIX_C_LANG_WIDE_CHAR
POSIX_DEVICE_IO
POSIX_DEVICE_SPECIFIC
POSIX_EVENT_MGMT
POSIX_FD_MGMT
POSIX_FIFO
POSIX_FILE_ATTRIBUTES
POSIX_FILE_LOCKING
POSIX_FILE_SYSTEM
POSIX_FILE_SYSTEM_EXT
POSIX_JOB_CONTROL
POSIX_MULTI_PROCESS
POSIX_NETWORKING

Table 9-1: POSIX.1 Units of Functionality Requirements (Continued)

Unit of Functionality
POSIX_PIPE
POSIX_REGEXP
POSIX_SHELL_FUNC
POSIX_SIGNALS
POSIX_SIGNAL_JUMP
POSIX_SINGLE_PROCESS
POSIX_STRING_MATCHING
POSIX_SYMBOLIC_LINKS
POSIX_SYSTEM_DATABASE
POSIX_THREADS_BASE
POSIX_USER_GROUPS
POSIX_WIDE_CHAR_IO
XSI_DYNAMIC_LINKING
XSI_SYSTEM_LOGGING
XSI_THREAD_MUTEX_EXT
XSI_THREADS_EXT

The Multi-Purpose Realtime System Profile implementation shall support the following options defined in POSIX.1, by defining the associated symbol with a value greater than zero:

Table 9-2: POSIX.1 Option Requirements

Option
_POSIX_ADVISORY_INFO
_POSIX_ASYNCHRONOUS_IO
_POSIX_CHOWN_RESTRICTED
_POSIX_CLOCK_SELECTION
_POSIX_CPUTIME
_POSIX_FSYNC
_POSIX_JOB_CONTROL
_POSIX_MAPPED_FILES
_POSIX_MEMLOCK
_POSIX_MEMLOCK_RANGE
_POSIX_MEMORY_PROTECTION
_POSIX_MESSAGE_PASSING
_POSIX_MONOTONIC_CLOCK
_POSIX_NO_TRUNC
_POSIX_PRIORITIZED_IO
_POSIX_PRIORITY_SCHEDULING
_POSIX_RAW_SOCKETS
_POSIX_REALTIME_SIGNALS
_POSIX_REGEXP
_POSIX_SAVED_IDS
_POSIX_SEMAPHORES
_POSIX_SHARED_MEMORY_OBJECTS

Table 9-2: POSIX.1 Option Requirements (Continued)

Option
_POSIX_SHELL
_POSIX_SPAWN
_POSIX_SPORADIC_SERVER
_POSIX_SYNCHRONIZED_IO
_POSIX_THREAD_ATTR_STACKADDR
_POSIX_THREAD_ATTR_STACKSIZE
_POSIX_THREAD_CPUTIME
_POSIX_THREAD_PRIO_INHERIT
_POSIX_THREAD_PRIO_PROTECT
_POSIX_THREAD_PRIORITY_SCHEDULING
_POSIX_THREAD_PROCESS_SHARED
_POSIX_THREAD_SAFE_FUNCTIONS
_POSIX_THREAD_SPORADIC_SERVER
_POSIX_TIMEOUTS
_POSIX_TIMERS
_POSIX_TRACE
_POSIX_TRACE_EVENT_FILTER
_POSIX_TRACE_LOG
_POSIX_VDISABLE

The type *off_t* shall be capable of storing any value contained in type *long*.

The minimum value of _POSIX_NGROUPS_MAX shall be at least 8.

The minimum value of CHILD_MAX shall be at least 25.

The value of _POSIX_TIMER_MAX shall be at least 64.

The value of _POSIX_RTSIG_MAX shall be at least 16.

The range of priorities associated with the SCHED_RR scheduling policy shall have at least 31 distinct values that are less than the maximum priority of the SCHED_FIFO policy.

An implementation conforming to PSE54 shall support the PTHREAD_SCOPE_SYSTEM scheduling contention scope. In addition, it may support PTHREAD_SCOPE_PROCESS. For a description of the scheduling contention scope see the System Interfaces volume of POSIX.1, Section 2.9.2, “Thread Scheduling”.

An implementation conforming to PSE54 shall provide a mechanism to configure the system so that the scheduling allocation domain has size one, and so that the binding of threads to scheduling allocation domains remains static. The mechanism by which this requirement is achieved shall be implementation defined. In addition, a PSE54 implementation may provide other configurations or facilities to change the size of the allocation domain and the bindings of threads to allocation domains. For a description of the scheduling allocation domain see the System Interfaces volume of POSIX.1, Section 2.9.2, “Thread Scheduling”.

9.2.2 POSIX.5c Requirements (Ada Language Option)

The Multi-Purpose Realtime System Profile implementation shall include interfaces as defined in POSIX.5c for the following units of functionality (see Table 1-2 through Table 1-18):

Table 9-3: POSIX.1 Units of Functionality Requirements

Unit of Functionality
POSIX_ADA_LANG_SUPPORT
POSIX_DEVICE_IO
POSIX_DEVICE_SPECIFIC
POSIX_EVENT_MGMT
POSIX_FD_MGMT
POSIX_FIFO
POSIX_FILE_ATTRIBUTES
POSIX_FILE_SYSTEM
POSIX_JOB_CONTROL
POSIX_MULTI_PROCESS
POSIX_NETWORKING
POSIX_PIPE
POSIX_SIGNALS
POSIX_SINGLE_PROCESS
POSIX_SYSTEM_DATABASE
POSIX_USER_GROUPS

The Multi-Purpose Realtime System Profile implementation shall support the following options defined in POSIX.5c, by defining the associated option subtypes to have the range `True..True`, with the exception of the Filename Truncation option for which the associated subtype shall have the range `False..False`:

Table 9-4: POSIX.5c Option Requirements

POSIX.5c Option
Asynchronous I/O
Change Owner Restriction
File Synchronization
Memory Mapped Files
Memory Locking
Memory Range Locking
Memory Protection
Message Queues
Filename Truncation
Prioritized I/O
Priority Process Scheduling
Realtime Signals
Saved IDs Support
Semaphores

Table 9-4: POSIX.5c Option Requirements (Continued)

POSIX.5c Option
Shared Memory Objects
Synchronized I/O
Mutexes Supported
Mutex Priority Inheritance
Mutex Priority Ceiling
Process Shared
Timers

The service `POSIX_Terminal_Functions.Disable_Control_Character` shall not raise `POSIX_Error` with an error code of `Operation_Not_Implemented`.

`POSIX_Limits.Child_Processes_Maxima'First` shall be at least 25.

`POSIX_Limits.Groups_Maxima'First` shall be at least 8.

`POSIX_Limits.Timers_Maxima'First` shall be at least 64.

`POSIX_Limits.Realtime_Signals_Maxima'First` shall be at least 16.

Regarding task priority scheduling, the implementation shall support the following requirements from POSIX.5c and the Ada95 RM:

— The implementation shall support the priority model defined in the Ada95 RM, clause D.1, and the pragmas and package interfaces defined in the Ada95 RM, clauses D.2-D.5.

— The implementation shall meet the requirements of POSIX.5c, section 13.3.1.

Implementations of the PSE54 profile shall support the `POSIX_Profiles` package defined in Annex A of this standard.

Subprograms not supported by a given profile shall raise `POSIX_Error`, returning an error code of `Operation_Not_Supported`, except as noted otherwise.

All `Image` and `Value` functions that appear in the packages supported by a profile must be implemented.

Where an overloaded subprogram is required by a unit of functionality, all forms of the subprogram appearing in the referenced clause must be supported, except as otherwise noted.

9.3 Application Constraints

The Multi-Purpose Realtime System profile defined in this standard requires only specific units of functionality of the required standards. The absence of particular elements of these standards introduces constraints on the use of some of the features of particular operations. This clause defines the constraints that an application strictly conforming to one of the profiles shall observe when using each of the operations required by that profile.

9.3.1 Constraints related to POSIX.1 Interfaces (C Language Option)

This profile has no constraints on the application related to POSIX.1 interfaces, because it requires the implementation to be POSIX.1 conforming.

9.3.2 Constraints related to POSIX.5c Interfaces (Ada Language Option)

An application strictly conforming to PSE54 shall not attempt to bind a signal to a task entry.

9.4 Shell and Utility Requirements

An implementation of the Multi-Purpose Realtime System Profile shall provide all the mandatory utilities in the Shell and Utilities volume of POSIX.1 with all the functional behavior described therein. The system shall support the Large File capabilities described in the Shell and Utilities volume of POSIX.1.

If the C Language Option is supported, the following options of the Shell and Utilities volume of POSIX.1 shall be supported:

Table 9-5: Shell and Utilities Option Requirements (C Language Option)

Option
POSIX2_C_BIND
POSIX2_CDEV
POSIX2_CHAR_TERM
POSIX2_FORT_RUN
POSIX2_SW_DEV
POSIX2_UPE

If the Ada Language Option is supported, the following options of the Shell and Utilities volume of POSIX.1 shall be supported:

Table 9-6: Shell and Utilities Option Requirements (Ada Language Option)

Option
POSIX2_CHAR_TERM
POSIX2_FORT_RUN
POSIX2_SW_DEV
POSIX2_UPE

9.5 Development Platform Requirements

One or more of the development options in 9.5.1 and 9.5.2 shall be implemented.

9.5.1 C Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this standard profile. This platform shall include the POSIX2_C_BIND, POSIX2_C_DEV, and POSIX2_SW_DEV options from the Shell and Utilities Volume of POSIX.1.

9.5.1.1 Option Indicator

The presence of the C Language Development Option shall be indicated by the symbol _POSIX_AEP_REALTIME_LANG_C99 being defined in the required header `<unistd.h>`. In addition, the presence of the C Language Development Option may be indicated by the subtype `POSIX_Profiles.Realtime_Lang_C99` having the range `True..True`.

9.5.2 Ada Language Development Option

If this option is provided, the implementor shall define a Development Platform and an environment capable of preparing for execution an application conformant with this profile including applicable portions of the following:

— The Ada95 RM {1}

— POSIX.5c {6}

— The POSIX2_SW_DEV option from the Shell and Utilities Volume of POSIX.1.

9.5.2.1 Option Indicator

The presence of the Ada Language Development Option shall be indicated by the subtype `POSIX_Profiles.Realtime_Lang_Ada95` having the range `True..True`. In addition, the presence of the Ada Language Development Option may be indicated by the symbol _POSIX_AEP_REALTIME_LANG_Ada95 being defined in the header `<unistd.h>`.

9.6 Rationale for Operating System Requirements (informative)

(This subclause is not a normative part of IEEE Std P1003.13)

9.6.1 Operating System Interface Requirements

This profile is based on existing practice in real-time systems that are built using general-purpose computers, such as workstations. These systems have general-purpose computing requirements such as a full featured file system, networking, virtual memory management, graphical user interfaces, multi-user access control, etc. In addition, they have real-time requirements, and thus the need for a real-time operating system that provides a full POSIX.1 implementation and also the realtime extensions described in this profile.

9.6.1.1 Process Primitives

The process control functions (which include process creation and execution) are the basic operating system services required to support multiple processes, and are therefore required by both realtime and non-realtime applications in these real-time systems.

9.6.1.2 Signals

Signal services are a basic mechanism within POSIX-based systems and are required for error and event handling. Realtime systems typically have several logically concurrent software elements executing. Each such entity must respond to several cyclic and/or acyclic stimuli, often in a time-critical manner. Although purely synchronous models can supply such functionality via the use of additional processes or threads, the current realtime practice for asynchronous notification for events such as timeout, message arrival, and hardware interrupt can generally be expected to offer higher performance and lower latency. Realtime Signals provide the reliable high-performance mechanism to support such notification.

The minimum number of realtime signals that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 8, up to 16. The rationale for this increase is that there are many applications that have more than 8 different kinds of events. Doubling the number of required realtime signals should have a minimum impact on the signal management overhead, while significantly increases the number of event kinds that can be used by a strictly conforming application.

9.6.1.3 Process Environment

The functions from the POSIX.1 Process Environment group are deemed necessary to allow an application to determine and configure its system environment. This allows a single version of an application to be run on similar but differing platforms.

Since the systems will require multiple processes and multiple users, and because they must support both commercial-off-the-shelf (COTS) and realtime applications, the entire set of ID functions is needed.

9.6.1.4 Files and Directories

All file and directory operations are required to support system applications and their filesystems. Although only a few of the path operation functions are required to support realtime activities, the whole set is required for systems that support COTS applications.

The Advisory Information option is required to allow the application to provide hints about the way in which is going to perform file operations, so that implementations can provide a better degree of timing predictability for those operations.

The File Locking option is required in the C-language option to maintain a consistent and safe way of accessing stdio (*FILE* *) objects from threads, across the four realtime profiles.

The File Descriptor Management unit of functionality is included to aid the handling of file descriptors across the process creation and program execution operations.

9.6.1.5 Input and Output Primitives

The functions contained in the Device I/O unit of functionality are required to support I/O on devices, files, and special files.

Although asynchronous I/O can be easily implemented using threads dedicated to I/O, it is required in the PSE54 profile to support portability of applications that may have been developed before POSIX threads implementations were widely available.

9.6.1.6 Synchronized Input and Output

These realtime systems that use file management systems will frequently require synchronized I/O to provide data integrity and/or relinquish resources to other processes. Synchronized I/O as defined in POSIX.1 provides these mechanisms.

9.6.1.7 Device- and Class-Specific Functions

The terminal control functions are required for systems to support COTS applications and for the standard terminal devices that may be attached to the computer system. To support non-standard terminal devices, additional functions may be necessary.

9.6.1.8 System Databases, Users and Groups

The group and user database access functions are required for COTS database applications that may require them.

9.6.1.9 Synchronization

Mutexes and Condition Variables are required as part of threads model of concurrency.

Semaphores are required to synchronize a signal handler with some other process or thread. Semaphores are also required to support portability of applications that might be using this mechanism instead of the preferred mutexes and condition variables. It must be noted, however, that POSIX semaphores do not have the mechanisms built in to avoid unbounded priority inversion when using them for mutually exclusive access to shared resources. Mutexes with the appropriate priority inheritance or priority ceiling (also called priority protection) protocols can be used to avoid this unbounded priority inversion. The Process Shared option is required to support applications requiring this mechanism for synchronization across different processes.

Barriers are not required because they can easily be implemented using mutexes and condition variables. Although a direct implementation of barriers can have a significant efficiency benefit in some multiprocessor architectures, a mutex-and-condition-variable implementation will not be significantly slower in most architectures, and thus requiring barriers for all implementations is not justified.

Spin locks are not required because, although they are an efficient synchronization mechanism, they cannot be portably used with the current POSIX.1 interfaces in realtime applications. If a realtime scheduling policy such as SCHED_FIFO or SCHED_RR is used, spin locks may cause deadlock on a single processor. On multiprocessors, to avoid deadlock, it would be necessary for threads using a given lock to be allocated to different processors. There are no standard APIs in the current POSIX.1 to allocate threads to specific processors.

Reader/Writer Locks are not required because they are not designed to avoid unbounded priority inversion, and thus very long delays could occur in realtime applications, with a low but nevertheless non-zero probability. It is expected that a future revision of the POSIX.1 standard will add the priority inheritance and/or priority ceiling options to reader/writer locks, which would eliminate the unbounded priority inversion.

9.6.1.10 Priority Scheduling

This realtime environment requires the ability to do scheduling of concurrent processes and threads with a preemptive priority-based scheduler to ensure that hard deadlines are met. Thread and process priority scheduling are required for realtime applications. The Sporadic Server Scheduling option is also required for processes and threads, to enhance support of applications with aperiodic timing requirements. The POSIX_PRIORITY_RANGES unit of functionality is not required because its functions are already included in the required _POSIX_PRIORITY_SCHEDULING option.

A common requirement of realtime systems is that they be able to run threads or processes with real-time requirements together with threads with no real-time requirements. One common way of doing this is by having the real-time threads run under the SCHED_FIFO scheduling policy, while the non real-time threads run at a lower priority under the round-robin policy (SCHED_RR) to fairly share the avail-

able portion of the processor among them. POSIX requires each policy to have a range of priorities of at least 32 distinct values, but does not impose any requirements on how these priority ranges relate to each other. It could happen that most or all of the SCHED_RR priorities were larger than the SCHED_FIFO priorities, thus making it impossible to mix realtime and non-realtime threads as required above. To solve this problem in a portable way, this profile requires that there are at least 31 SCHED_RR priority levels below the maximum priority of SCHED_FIFO. In this way, a strictly conforming application can use the inclusive priority range [*max_FIFO_prio*, *max_FIFO_prio*-30] with SCHED_FIFO for real-time threads (with a total of 31 priority levels), and then use the priority value min(*max_FIFO_prio*-31,*max_RR_prio*) with the SCHED_RR policy, for the non real-time threads, with guarantee that the latter priority value is valid for the round-robin policy.

The implementation is required to support the PTHREAD_SCOPE_SYSTEM thread-scheduling contention scope. The contention scope of a thread defines the set of threads with which the thread competes for use of the processing resources. A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends for resources with all other threads in the system that have the same scheduling allocation domain. This allows a consistent scheduling of threads across the system and therefore a predictable timing behavior. As a consequence, this is the preferred method for realtime systems.

The current POSIX.1 specification allows implementations to support either system-wide or process-wide contention scope, or both. This represents a compromise that tries to address the requirements of both realtime and non-realtime applications, but introduces a potential source for non portability. Because the realtime profiles are specifically targeted at realtime systems, the system-wide contention scope option is required in the profiles that support multiple processes. Process-wide contention scope may also be provided, perhaps for the non realtime threads of the application.

Support for a scheduling allocation domain of size one and static binding of threads to allocation domains is required in all the realtime profiles to achieve predictable scheduling behavior. The allocation domain of a thread is the set of processors on which that thread can be scheduled at any given time. The POSIX.1 standard specifies that the scheduling rules have predictable effects only if the allocation domain is of size one; hence the need for this requirement. For single-processor systems the allocation domain is generally of size one and thus the application can meet the requirement just by specifying in the conformance document that the scheduling allocation domain is of size one and that static binding of threads to allocation domains is the default behavior.

9.6.1.11 Process Memory Locking

Realtime processes must be able to guarantee memory residency to reduce the latency for instruction fetches, data access, I/O operations, etc. The mechanism de-

scribed in the POSIX.1 Process Memory Locking extension will satisfy this requirement.

9.6.1.12 Shared Memory

The ability to share large volumes of data among many cooperating execution streams is required. The POSIX.1 Shared Memory extension provides this capability. Memory Mapped I/O may be implemented using the Shared Memory facility. An implementation must provide facilities for creating a block of physical memory in which the application may place devices and facilities for binding to a user-provided pathname through which a device may subsequently be opened as a Shared Memory special file, and mapped into the process address space for the purpose of performing I/O or other functions from applications programs.

The Memory Mapped Files is required because the implementation has file-system capabilities, and memory-mapped files are a convenient paradigm for reading and writing information in applications following this profile. In memory-mapped files, data can be manipulated as memory, and I/O data movement can be significantly reduced. The implementation of memory-mapped files does not require a significant amount of additional memory or execution overhead to achieve the additional capability.

System vendors are expected to implement the chosen interface in a manner that meets the needs of the applications. In particular, a rotating media-based implementation is not required by the interface definition.

Typed Memory objects are not required because they are useful only to systems with special hardware architectures that have various often specialized kinds of memory. Implementors providing support for such special architectures always have the option to provide typed memory objects as an extension.

9.6.1.13 Clocks and Timers

High-resolution timer functions are required in most realtime systems for implementing time management operations such as periodic activations, short duration time-outs, etc. The normal POSIX.1 time management functions *sleep*() and *alarm*() only provide a time resolution of one second, but many realtime systems require finer resolution for specifying time.

The Monotonic Clock is required for realtime applications to ensure that deadlines and timing requirements are not affected by clock jumps.

The Clock Selection option is required to enable choosing the clock on which sleep operations are performed, and to have access to an absolute sleep operation, which is a common requirement in realtime applications with periodic timing requirements.

CPU-Time clocks and timers are required as a means to detect and handle situations in which a thread overruns its assigned maximum execution time. Delimiting the execution times of the different threads in the application provides temporal partitioning in realtime applications, and thus increases predictability and reliability.

The Timeouts option is a general requirement for realtime applications and thus is required in this profile.

The minimum number of per-process timers that the implementation is required to support has been increased from the number specified in the POSIX.1 standard, 32, up to 64, which is the required minimum number of threads per process. The reason for this increase is that there are many applications that require one timer per thread (either realtime or CPU-time based).

9.6.1.14 Message Passing

These realtime systems typically include some form of message queuing mechanism for communication among processes or threads. The POSIX.1 message passing offers an appropriate level of performance to provide this functionality.

9.6.1.15 Threads

The basic assumption in this profile is that the system will consist of one or more processes with multiple threads. Therefore, all thread services are required. The POSIX_THREADS_BASE unit of functionality was specified in this document instead of the _POSIX_THREADS option, because this option requires reader/writer locks, but this profile does not.

9.6.1.16 Tracing

Tracing is required for the PSE54 environment because it provides an excellent mechanism to support post-failure analysis, particularly for failures having a low probability of occurrence.

The Trace Event Filtering option is required for the system to be able to filter out those trace events that are not meaningful for the application, thus making better use of system resources by capturing only the interesting events.

Because the PSE54 profile requires general file system capabilities, the Trace Log option is required for this profile.

9.6.1.17 Networking

Today, virtually all of the platforms and applications belonging to the PSE54 environment require network communications, and thus the networking unit of functionality is required in this profile. The Raw Sockets option is required to aid reconfiguration of networked applications, and to implement special protocols directly, without the weight of a full protocol stack. The Internet Protocol Version 6 option is not required because most applications are not using this version of the protocol yet.

9.6.1.18 Event Management

The *select*() function is usually associated with networking facilities, which are required for PSE54, and thus the Event Management unit of functionality is required in the PSE54 environment.

9.6.1.19 Interfaces Related to the Shell and Utilities

The interfaces defined in the POSIX_REGEXP and POSIX_SHELL_FUNC are required in PSE54 environments, because of their general-purpose computing requirements.

9.6.1.20 X/Open Units of Functionality and Options

Some XSI Units of Functionality (XSI_C_LANG_SUPPORT, XSI_DEVICE_IO, XSI_DEVICE_SPECIFIC, XSI_FD_MGMT, XSI_FILE_SYSTEM, XSI_IPC, XSI_JOB_CONTROL, XSI_JUMP, XSI_MATH, XSI_MULTI_PROCESS, XSI_SIGNALS, XSI_SINGLE_PROCESS, XSI_SYSTEM_DATABASE, XSI_TIMERS, XSI_USER_GROUPS, XSI_WIDE_CHAR) have interfaces that represent extensions or alternatives to interfaces in other Units of Functionality or POSIX.1 options, and therefore are not necessary for PSE54 environments.

The XSI_DBM unit of functionality includes interfaces for database management that are not required in the PSE54 application environment.

The XSI_DYNAMIC_LINKING unit of functionality is required for PSE54 systems, which usually execute a mixture of realtime and non realtime activities in a typically dynamic context.

The XSI_I18N unit of functionality provides facilities for natural language messages to the user, which are not required all PSE54 systems. It remains as an optional feature.

The XSI_SYSTEM_LOGGING unit of functionality provides facilities for logging system activities, which are usually required in PSE54 environments. Therefore, this unit of functionality is required.

The XSI_THREAD_MUTEX_EXT unit of functionality is required because it has options for controlling the behavior of mutexes under erroneous application use. This capability is interesting for any realtime application, including those targeted at small embedded systems.

The XSI_THREADS_EXT unit of functionality is required because it provides functions to better control a thread's stack. This is considered useful for any realtime application.

The _XOPEN_CRYPT option provides cryptography facilities that are not required in all PSE54 environments. It remains as an optional feature.

The _XOPEN_LEGACY option provides facilities for backwards compatibility that are not required in most PSE54 environments.

The _XOPEN_STREAMS option provides facilities that are not required in most PSE54 environments.

9.6.1.21 Language-Specific Services for the C Programming Language

Full support for the C Language standard is required in the C language option.

9.6.1.22 Language-Specific Services for the Ada Programming Language

Support for the Ada language-specific services defined in POSIX.5c is required in the Ada language option.

9.6.2 Shell and Utility Requirements

The utilities and facilities described in the Shell and Utilities Volume of POSIX.1 are required in PSE54 environments.

9.6.3 Development Platform Requirements

The implementation is required to define a development environment in which a PSE54 application can be prepared for execution on the target platform. For this

profile, in most cases the development and the target platform roles will be combined in the same system.

Annex A: POSIX Profiles Package (Ada Language)

(Normative)

The package `POSIX_Profiles` shall be supported by all profiles. The `Boolean` subtypes contained in this package shall indicate the profiles and options supported by the implementation. Supported profiles and options shall be indicated by the appropriate identifier having the range `True..True`; unsupported profiles and options shall have the range `False..False`.

```
package POSIX_Profiles is

  -- Profile options
  subtype Realtime_Minimal    is Boolean range <Implementation Defined>;
  subtype Realtime_Controller is Boolean range <Implementation Defined>;
  subtype Realtime_Dedicated  is Boolean range <Implementation Defined>;
  subtype Realtime_Multi      is Boolean range <Implementation Defined>;

  -- Language development options
  subtype Realtime_Lang_C99   is Boolean range <Implementation Defined>;
  subtype Realtime_Lang_Ada95 is Boolean range <Implementation Defined>;

end POSIX_Profiles;
```

Annex B: Description of Optional Interfaces

(Informative)

B.1 POSIX.1 Options

The following table shows the functions included under each of the options specified in the System Interfaces volume of POSIX.1. Each row of this table contains all the functions included under the first named option, and also under combinations of that option with other options.

Table B-1: Functions under each POSIX.1 System Interface Option

_POSIX_ADVISORY_INFO *posix_fadvise*(), *posix_fallocate*(), *posix_memalign*() **_POSIX_ADVISORY_INFO and either _POSIX_MAPPED_FILES or _POSIX_SHARED_MEMORY_OBJECTS** *posix_madvise*()
_POSIX_ASYNCHRONOUS_IO *aio_cancel*(), *aio_error*(), *aio_fsync*(), *aio_read*(), *aio_return*(), *aio_suspend*(), *aio_write*(), *lio_listio*()
_POSIX_BARRIERS and _POSIX_THREADS *pthread_barrier_destroy*(), *pthread_barrier_init*(), *pthread_barrier_wait*(), *pthread_barrierattr_destroy*(), *pthread_barrierattr_init*(), **POSIX_BARRIERS, _POSIX_THREADS and _POSIX_THREAD_PROCESS_SHARED** *pthread_barrierattr_getpshared*(), *pthread_barrierattr_setpshared*()
_POSIX_CHOWN_RESTRICTED No functions under this option
_POSIX_CLOCK_SELECTION *clock_nanosleep*() **_POSIX_CLOCK_SELECTION and _POSIX_THREADS** *pthread_condattr_getclock*(), *pthread_condattr_setclock*()
_POSIX_CPUTIME *clock_getcpuclokid*()

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_POSIX_FSYNC *fsync*()
_POSIX_IPV6 No functions under this option
_POSIX_JOB_CONTROL See the POSIX_JOB_CONTROL unit of functionality
_POSIX_MAPPED_FILES or _POSIX_SHARED_MEMORY_OBJECTS *mmap*(), *munmap*() **_POSIX_MAPPED_FILES and _POSIX_SYNCHRONIZED_IO** *msync*() **_POSIX_MAPPED_FILES and _POSIX_ADVISORY_INFO** *posix_madvise*()
_POSIX_MEMLOCK *mlockall*(), *munlockall*()
_POSIX_MEMLOCK_RANGE *mlock*(), *munlock*()
_POSIX_MEMORY_PROTECTION *mprotect*()
_POSIX_MESSAGE_PASSING *mq_close*(), *mq_getattr*(), *mq_notify*(), *mq_open*(), *mq_receive*(), *mq_send*(), *mq_setattr*(), *mq_unlink*(), **_POSIX_MESSAGE_PASSING and _POSIX_TIMEOUTS** *mq_timedreceive*(), *mq_timedsend*()
_POSIX_MONOTONIC_CLOCK No functions under this option
_POSIX_NO_TRUNC No functions under this option
_POSIX_PRIORITIZED_IO No functions under this option
_POSIX_PRIORITY_SCHEDULING *sched_get_priority_max*(), *sched_get_priority_min*(), *sched_getparam*(), *sched_getscheduler*(), *sched_rr_get_interval*(), *sched_setparam*(), *sched_setscheduler*() **_POSIX_PRIORITY_SCHEDULING or _POSIX_THREADS** *sched_yield*(), **_POSIX_PRIORITY_SCHEDULING and _POSIX_SPAWN** *posix_spawnattr_getschedparam*(), *posix_spawnattr_setschedparam*(), *posix_spawnattr_getschedpolicy*(), *posix_spawnattr_setschedpolicy*()

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_POSIX_RAW_SOCKETS No functions under this option
_POSIX_READER_WRITER_LOCKS See the POSIX_RW_LOCKS unit of functionality
_POSIX_REALTIME_SIGNALS *sigqueue*(), *sigtimedwait*(), *sigwaitinfo*()
_POSIX_REGEXP See POSIX_REGEXP unit of functionality.
_POSIX_SAVED_IDS No functions under this option
_POSIX_SEMAPHORES *sem_close*(), *sem_destroy*(), *sem_getvalue*(), *sem_init*(), *sem_open*(), *sem_post*(), *sem_trywait*(), *sem_wait*(), *sem_unlink*() **_POSIX_SEMAPHORES and _POSIX_TIMEOUTS** *sem_timedwait*()
_POSIX_SHARED_MEMORY_OBJECTS *shm_open*(), *shm_unlink*() **_POSIX_SHARED_MEMORY_OBJECTS and _POSIX_ADVISORY_INFO** *posix_madvise*() **_POSIX_SHARED_MEMORY_OBJECTS or _POSIX_MAPPED_FILES** *mmap*(), *munmap*()
_POSIX_SPAWN *posix_spawn*(), *posix_spawn_file_actions_addclose*(), *posix_spawn_file_actions_adddup2*(), *posix_spawn_file_actions_addopen*(), *posix_spawn_file_actions_destroy*(), *posix_spawn_file_actions_init*(), *posix_spawnattr_destroy*(), *posix_spawnattr_getflags*(), *posix_spawnattr_getpgroup*(), *posix_spawnattr_getsigdefault*(), *posix_spawnattr_getsigmask*(), *posix_spawnattr_init*(), *posix_spawnattr_setflags*(), *posix_spawnattr_setpgroup*(), *posix_spawnattr_setsigdefault*(), *posix_spawnattr_setsigmask*(), *posix_spawnp*() **_POSIX_SPAWN and _POSIX_PRIORITY_SCHEDULING** *posix_spawnattr_getschedparam*(), *posix_spawnattr_setschedparam*(), *posix_spawnattr_getschedpolicy*(), *posix_spawnattr_setschedpolicy*()
_POSIX_SPIN_LOCKS and _POSIX_THREADS *pthread_spin_destroy*(), *pthread_spin_init*(), *pthread_spin_lock*(), *pthread_spin_trylock*(), *pthread_spin_unlock*()
_POSIX_SPORADIC_SERVER No functions under this option

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_POSIX_SYNCHRONIZED_IO *fdatasync*()
_POSIX_SYNCHRONIZED_IO and _POSIX_MAPPED_FILES *msync*()
_POSIX_THREAD_ATTR_STACKADDR and _POSIX_THREADS *pthread_attr_getstackaddr*(), *pthread_attr_setstackaddr*()
_POSIX_THREAD_ATTR_STACKADDR, _POSIX_THREADS and _POSIX_THREAD_ATTR_STACKSIZE *pthread_attr_getstack*(), *pthread_attr_setstack*()
_POSIX_THREAD_ATTR_STACKSIZE and _POSIX_THREADS *pthread_attr_getstacksize*(), *pthread_attr_setstacksize*()[a]
_POSIX_THREAD_ATTR_STACKSIZE, _POSIX_THREADS and _POSIX_THREAD_ATTR_STACKADDR *pthread_attr_getstack*(), *pthread_attr_setstack*()
_POSIX_THREAD_CPUTIME and _POSIX_THREADS *pthread_getcpuclockid*()
_POSIX_THREAD_PRIO_INHERIT and _POSIX_THREADS *pthread_mutexattr_getprotocol*(), *pthread_mutexattr_setprotocol*()
_POSIX_THREAD_PRIO_PROTECT and _POSIX_THREADS *pthread_mutex_getprioceiling*(), *pthread_mutex_setprioceiling*(), *pthread_mutexattr_getprioceiling*(), *pthread_mutexattr_getprotocol*(), *pthread_mutexattr_setprioceiling*(), *pthread_mutexattr_setprotocol*()
_POSIX_THREAD_PRIORITY_SCHEDULING and _POSIX_THREADS *pthread_attr_getinheritsched*(), *pthread_attr_getschedpolicy*(), *pthread_attr_getscope*(), *pthread_attr_setinheritsched*(), *pthread_attr_setschedpolicy*(), *pthread_attr_setscope*(), *pthread_getschedparam*(), *pthread_setschedparam*(), *pthread_setschedprio*()
_POSIX_THREAD_PROCESS_SHARED and _POSIX_THREADS *pthread_condattr_getpshared*(), *pthread_condattr_setpshared*(), *pthread_mutexattr_getpshared*(), *pthread_mutexattr_setpshared*()
_POSIX_THREAD_PROCESS_SHARED, _POSIX_BARRIERS and _POSIX_THREADS *pthread_barrierattr_getpshared*(), *pthread_barrierattr_setpshared*()
_POSIX_THREAD_PROCESS_SHARED, _POSIX_READER_WRITER_LOCKS and _POSIX_THREADS *pthread_rwlockattr_getpshared*(), *pthread_rwlockattr_setpshared*()
_POSIX_THREAD_SAFE_FUNCTIONS *asctime_r*(), *ctime_r*(), *flockfile*(), *ftrylockfile*(), *funlockfile*(), *getc_unlocked*(), *getchar_unlocked*(), *getgrgid_r*(), *getgrnam_r*(), *getlogin_r*(), *getpwnam_r*(), *getpwuid_r*(), *gmtime_r*(), *localtime_r*(), *putc_unlocked*(), *putchar_unlocked*(), *rand_r*(), *readdir_r*(), *strerror_r*(), *strtok_r*(), *ttyname_r*()

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_POSIX_THREAD_SPORADIC_SERVER No functions under this option
_POSIX_THREADS *pthread_atfork*(), *pthread_attr_destroy*(), *pthread_attr_getdetachstate*(), *pthread_attr_getschedparam*(), *pthread_attr_init*(), *pthread_attr_setdetachstate*(), *pthread_attr_setschedparam*(), *pthread_cancel*(), *pthread_cleanup_pop*(), *pthread_cleanup_push*(), *pthread_cond_broadcast*(), *pthread_cond_destroy*(), *pthread_cond_init*(), *pthread_cond_signal*(), *pthread_cond_timedwait*(), *pthread_cond_wait*(), *pthread_condattr_destroy*(), *pthread_condattr_init*(), *pthread_create*(), *pthread_detach*(), *pthread_equal*(), *pthread_exit*(), *pthread_getspecific*(), *pthread_join*(), *pthread_key_create*(), *pthread_key_delete*(), *pthread_kill*(), *pthread_mutex_destroy*(), *pthread_mutex_init*(), *pthread_mutex_lock*(), *pthread_mutex_trylock*(), *pthread_mutex_unlock*(), *pthread_mutexattr_destroy*(), *pthread_mutexattr_init*(), *pthread_once*(), *pthread_self*(), *pthread_setcalcelstate*(), *pthread_setcanceltype*(), *pthread_setspecific*(), *pthread_sigmask*(), *pthread_testcancel*() **_POSIX_THREADS and _POSIX_CLOCK_SELECTION** *pthread_condattr_getclock*(), *pthread_condattr_setclock*() **_POSIX_THREADS and _POSIX_BARRIERS** *pthread_barrier_destroy*(), *pthread_barrier_init*(), *pthread_barrier_wait*(), *pthread_barrierattr_destroy*(), *pthread_barrierattr_init*(), **_POSIX_THREADS, POSIX_BARRIERS and _POSIX_THREAD_PROCESS_SHARED** *pthread_barrierattr_getpshared*(), *pthread_barrierattr_setpshared*() **_POSIX_THREADS and _POSIX_SPIN_LOCKS** *pthread_spin_destroy*(), *pthread_spin_init*(), *pthread_spin_lock*(), *pthread_spin_trylock*(), *pthread_spin_unlock*() **_POSIX_THREADS and _POSIX_THREAD_ATTR_STACKADDR** *pthread_attr_getstackaddr*(), *pthread_attr_setstackaddr*() **_POSIX_THREADS, _POSIX_THREAD_ATTR_STACKADDR and _POSIX_THREAD_ATTR_STACKSIZE** *pthread_attr_getstack*(), *pthread_attr_setstack*() **_POSIX_THREADS and _POSIX_THREAD_ATTR_STACKSIZE** *pthread_attr_getstacksize*(), *pthread_attr_setstacksize*()[a] **_POSIX_THREADS and _POSIX_THREAD_CPUTIME** *pthread_getcpuclockid*() **_POSIX_THREADS and either _POSIX_THREAD_PRIO_INHERIT or _POSIX_THREAD_PRIO_PROTECT** *pthread_mutexattr_getprotocol*(), *pthread_mutexattr_setprotocol*() *This table row continued on next page...*

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

...table row continued from previous page
_POSIX_THREAD_PRIO_PROTECT and _POSIX_THREADS *pthread_mutex_getprioceiling*(), *pthread_mutex_setprioceiling*(), *pthread_mutexattr_getprioceiling*(), *pthread_mutexattr_setprioceiling*()
_POSIX_THREADS and _POSIX_THREAD_PRIORITY_SCHEDULING *pthread_attr_getinheritsched*(), *pthread_attr_getschedpolicy*(), *pthread_attr_getscope*(), *pthread_attr_setinheritsched*(), *pthread_attr_setschedpolicy*(), *pthread_attr_setscope*(), *pthread_getschedparam*(), *pthread_setschedparam*(), *pthread_setschedprio*()
_POSIX_THREADS and _POSIX_THREAD_PROCESS_SHARED *pthread_condattr_getpshared*(), *pthread_condattr_setpshared*(), *pthread_mutexattr_getpshared*(), *pthread_mutexattr_setpshared*(),
_POSIX_THREADS, _POSIX_THREAD_PROCESS_SHARED and _POSIX_READER_WRITER_LOCKS *pthread_rwlockattr_getpshared*(), *pthread_rwlockattr_setpshared*()
_POSIX_THREADS and POSIX_TIMEOUTS *pthread_mutex_timedlock*()
_POSIX_THREADS, _POSIX_TIMEOUTS and _POSIX_READER_WRITER_LOCKS *pthread_rwlock_timedrdlock*(), *pthread_rwlock_timedwrlock*()
_POSIX_THREADS and _POSIX_READER_WRITER_LOCKS *pthread_rwlock_destroy*(), *pthread_rwlock_init*(), *pthread_rwlock_rdlock*(), *pthread_rwlock_tryrdlock*(), *pthread_rwlock_trywrlock*(), *pthread_rwlock_unlock*(), *pthread_rwlock_wrlock*(), *pthread_rwlockattr_destroy*(), *pthread_rwlockattr_init*()
_POSIX_THREADS or _POSIX_PRIORITY_SCHEDULING *sched_yield*()
_POSIX_TIMEOUTS and _POSIX_MESSAGE_PASSING *mq_timedreceive*(), *mq_timedsend*()
_POSIX_TIMEOUTS, _POSIX_THREADS, and _POSIX_READER_WRITER_LOCKS *pthread_rwlock_timedrdlock*(), *pthread_rwlock_timedwrlock*()
_POSIX_TIMEOUTS and _POSIX_SEMAPHORES *sem_timedwait*()
_POSIX_TIMEOUTS and _POSIX_THREADS *pthread_mutex_timedlock*()
_POSIX_TIMEOUTS and _POSIX_TRACE *posix_trace_timedgetnext_event*()
_POSIX_TIMERS *clock_getres*(), *clock_gettime*(), *clock_settime*(), *nanosleep*(), *timer_create*(), *timer_detele*(), *timer_getoverrun*(), *timer_gettime*(), *timer_settime*()

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_POSIX_TRACE *posix_trace_attr_destroy*(), *posix_trace_attr_getclockres*(), *posix_trace_attr_getcreatetime*(), *posix_trace_attr_getgenversion*(), *posix_trace_attr_getname*(), *posix_trace_attr_getstreamfullpolicy*(), *posix_trace_attr_getmaxdatasize*(), *posix_trace_attr_getmaxsystemeventsize*(), *posix_trace_attr_getmaxusereventsize*(), *posix_trace_attr_getstreamsize*(), *posix_trace_attr_init*(), *posix_trace_attr_setname*(), *posix_trace_attr_setstreamfullpolicy*(), *posix_trace_attr_setmaxdatasize*(), *posix_trace_attr_setstreamsize*(), *posix_trace_clear*(), *posix_trace_create*(), *posix_trace_event*(), *posix_trace_eventid_open*(), *posix_trace_eventid_equal*(), *posix_trace_eventid_get_name*(), *posix_trace_eventtypelist_getnext_id*(), *posix_trace_eventtypelist_rewind*(), *posix_trace_get_attr*(), *posix_trace_get_status*(), *posix_trace_getnext_event*(), *posix_trace_shutdown*(), *posix_trace_start*(), *posix_trace_stop*(), *posix_trace_trygetnext_event*()
_POSIX_TRACE and _POSIX_TIMEOUTS *posix_trace_timedgetnext_event*()
_POSIX_TRACE and _POSIX_TRACE_INHERIT *posix_trace_attr_getinherited*(), *posix_trace_attr_setinherited*()
_POSIX_TRACE and _POSIX_TRACE_LOG *posix_trace_attr_getlogfullpolicy*(), *posix_trace_attr_getlogsize*(), *posix_trace_attr_setlogfullpolicy*(), *posix_trace_attr_setlogsize*(), *posix_trace_close*(), *posix_trace_open*(), *posix_trace_rewind*(), *posix_trace_create_withlog*(), *posix_trace_flush*()
_POSIX_TRACE and _POSIX_TRACE_EVENT_FILTER *posix_trace_eventset_add*(), *posix_trace_eventset_del*(), *posix_trace_eventset_empty*(), *posix_trace_eventset_fill*(), *posix_trace_eventset_ismember*(), *posix_trace_get_filter*(), *posix_trace_set_filter*(), *posix_trace_trid_eventid_open*()
_POSIX_TRACE_EVENT_FILTER and _POSIX_TRACE *posix_trace_eventset_add*(), *posix_trace_eventset_del*(), *posix_trace_eventset_empty*(), *posix_trace_eventset_fill*(), *posix_trace_eventset_ismember*(), *posix_trace_get_filter*(), *posix_trace_set_filter*(), *posix_trace_trid_eventid_open*()
_POSIX_TRACE_INHERIT and _POSIX_TRACE *posix_trace_attr_getinherited*(), *posix_trace_attr_setinherited*()
_POSIX_TRACE_LOG and _POSIX_TRACE *posix_trace_attr_getlogfullpolicy*(), *posix_trace_attr_getlogsize*(), *posix_trace_attr_setlogfullpolicy*(), *posix_trace_attr_setlogsize*(), *posix_trace_close*(), *posix_trace_open*(), *posix_trace_rewind*(), *posix_trace_create_withlog*(), *posix_trace_flush*()
_POSIX_TYPED_MEMORY_OBJECTS *posix_mem_offset*(), *posix_typed_mem_get_info*(), *posix_typed_mem_open*()
_POSIX_VDISABLE No functions under this option

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_XOPEN_CRYPT *crypt*(), *encrypt*(), *setkey*()
_XOPEN_ENH_I18N No functions under this option
_XOPEN_LEGACY *bcmp*(), *bcopy*(), *bzero*(), *ecvt*(), *fcvt*(), *ftime*(), *gcvt*(), *getwd*(), *index*(), *mktemp*(), *rindex*(), *utimes*(), *wcswcs*()
_XOPEN_REALTIME This Option Group consists of the set of the following options from within POSIX.1: _POSIX_ASYNCHRONOUS_IO _POSIX_FSYNC _POSIX_MAPPED_FILES _POSIX_MEMLOCK _POSIX_MEMLOCK_RANGE _POSIX_MEMORY_PROTECTION _POSIX_MESSAGE_PASSING _POSIX_PRIORITIZED_IO _POSIX_PRIORITY_SCHEDULING _POSIX_REALTIME_SIGNALS _POSIX_SEMAPHORES _POSIX_SHARED_MEMORY_OBJECTS _POSIX_SYNCHRONIZED_IO _POSIX_TIMERS
_XOPEN_REALTIME_THREADS This Option Group consists of the set of the following options from within POSIX.1: _POSIX_THREAD_PRIO_INHERIT _POSIX_THREAD_PRIO_PROTECT _POSIX_THREAD_PRIORITY_SCHEDULING
_XOPEN_SHM This option is included in the XSI_IPC unit of functionality
_XOPEN_STREAMS *fattach*(), *fdetach*(), *getmsg*(), *getpmsg*(), *ioctl*(), *isastream*(), *putmsg*(), *putpmsg*(),

Table B-1: Functions under each POSIX.1 System Interface Option (Continued)

_XOPEN_UNIX This Option Group consists of the functions included in the following units of functionality: XSI_C_LANG_SUPPORT XSI_DBM XSI_DEVICE_IO XSI_DEVICE_SPECIFIC XSI_DYNAMIC_LINKING XSI_FD_MGMT XSI_FILE_SYSTEM XSI_I18N XSI_IPC XSI_JOB_CONTROL XSI_JUMP XSI_MATH XSI_MULTI_PROCESS XSI_SIGNALS XSI_SINGLE_PROCESS XSI_SYSTEM_DATABASE XSI_SYSTEM_LOGGING XSI_THREAD_MUTEX_EXT XSI_THREADS_EXT XSI_TIMERS XSI_USER_GROUPS XSI_WIDE_CHAR

a. The *pthread_attr_getstacksize*() and *pthread_attr_setstacksize*() functions are wrongly listed under the _POSIX_THREAD_STACK_ADDRESS option in POSIX.1, but should be under the _POSIX_THREAD_STACK_SIZE option.

The following table shows the utilities included under each of the options specified in the Shell and Utilities volume of POSIX.1:

Table B-2: Utilities under each POSIX.1 Shell and Utilities Option

_POSIX_SHELL `sh`
_POSIX_C_BIND No utilities under this option
_POSIX2_C_DEV `c99`, `lex`, `yacc`
_POSIX2_CHAR_TERM No utilities under this option

Table B-2: Utilities under each POSIX.1 Shell and Utilities Option

_POSIX2_FORT_DEV `fort77`
_POSIX2_FORT_RUN `asa`
_POSIX2_LOCALEDEF No utilities under this option
_POSIX2_PBS `qalter`, `qdel`, `qhold`, `qmove`, `qmsg`, `qrerun`, `qrls`, `qselect`, `qsig`, `qstat`, `qsub`
_POSIX2_PBS_ACCOUNTING No utilities under this option.
_POSIX2_PBS_CHECKPOINT No utilities under this option.
_POSIX2_PBS_LOCATE No utilities under this option.
_POSIX2_PBS_MESSAGE No utilities under this option.
_POSIX2_PBS_TRACK No utilities under this option.

_POSIX2_SW_DEV
`ar`, `make`, `strip`
_POSIX2_SW_DEV and _POSIX2_UPE
`nm`

_POSIX2_UPE
`alias`, `at`, `batch`, `bg`, `command`, `crontab`, `csplit`, `ctags`, `df`, `du`, `ex`, `expand`, `fc`, `fg`, `file`, `jobs`, `mesg`, `more`, `newgrp`, `nice`, `patch`, `ps`, `renice`, `split`, `strings`, `tabs`, `talk`, `time`, `tput`, `unalias`, `unexpand`, `uudecode`, `uuencode`, `vi`, `who`, `write`
_POSIX2_UPE and _POSIX2_SW_DEV
`nm`

B.2 POSIX.5c Options

The following table shows the subprograms included under each of the options specified in POSIX.5c:

Table B-3: Packages and Subprograms under each POSIX.5c Option

Package	Subprogram
Asynchronous I/O	
POSIX_Asynchonous_IO	All except the two subprograms below
Asynchronous I/O and Synchronized I/O	
POSIX_Asynchonous_IO	`Synchronize_File` `Synchronize_Data`
Change Owner Restriction	None
File Synchronization	
POSIX_IO	`Synchronize_File`
Filename Truncation	None
Memory Mapped Files or Shared Memory Objects	
POSIX_IO	`Change_Permissions` `Truncate_File`
POSIX_Memory_Mapping	`Map_Memory`[a] `Unmap_Memory`
Memory Mapped Files and Synchronized I/O	
POSIX_Memory_Mapping	`Synchronize_Memory`
Memory Locking	
POSIX_Memory_Locking	All
Memory Protection	
POSIX_Memory_Mapping	`Change_Protection`
Memory Range Locking	
POSIX_Memory_Range_Locking	All
Message Queues	
POSIX_Message_Queues	All

Table B-3: Packages and Subprograms under each POSIX.5c Option (Continued)

Package	Subprogram
Mutexes	
POSIX_Mutexes	All except the subprograms below
POSIX_Condition_Variables	All except the subprograms below
Mutexes and Process Shared	
POSIX_Mutexes	Get_Process_Shared
	Set_Process_Shared
POSIX_Condition_Variables	Get_Process_Shared
	Set_Process_Shared
Mutexes and MutexPriority Ceiling	
POSIX_Mutexes	Set_Ceiling_Priority[a]
	Get_Ceiling_Priority[a]
Mutexes and either Mutex Priority Inheritance or MutexPriority Ceiling	
POSIX_Mutexes	Set_Locking_Policy
	Get_Locking_Policy
Mutex Priority Ceiling and Mutexes	
POSIX_Mutexes	Set_Ceiling_Priority[a]
	Get_Ceiling_Priority[a]
	Set_Locking_Policy
	Get_Locking_Policy
Mutex Priority Inheritance and Mutexes	
POSIX_Mutexes	Set_Locking_Policy
	Get_Locking_Policy
Network Management and Sockets Detailed Network Interface	
POSIX_Sockets	Set_Flags
	Get_Flags
	Set_Family
	Get_Family
	Set_Socket_Type
	Get_Socket_Type
	Set_Protocol_Number
	Get_Protocol_Number
	Get_Canonical_Name
	Get_Socket_Address_Info
	Get_Socket_Address_Info
	For_Every_Item
Poll	
POSIX_Event_Management	Get_File
	Set_File
	Get_Events
	Set_Events
	Get_Returned_Events
	Set_Returned_Events
	Poll
Prioritized I/O	None

Table B-3: Packages and Subprograms under each POSIX.5c Option (Continued)

Package	Subprogram
Priority Process Scheduling	
POSIX_Process_Scheduling	All
Process Shared and Mutexes	
POSIX_Mutexes	Get_Process_Shared Set_Process_Shared
POSIX_Condition_Variables	Get_Process_Shared Set_Process_Shared
Realtime Signals	
POSIX_Signals	Enable_Queueing Disable_Queueing Await_Signal[b] Await_Signal_Or_Timeout[b] Queue_Signal
Saved IDs Support	None
Select	
POSIX_Event_Management	Add Remove In_Set Select_File[a]
Semaphores	
POSIX_Semaphores	All
Shared Memory Objects	
POSIX_Shared_Memory_Objects	All
POSIX_Generic_Shared_Memory	All
Shared Memory Objects and Memory Range Locking	
POSIX_Generic_Shared_Memory	Lock_Shared_Memory Unlock_Shared_Memory
Shared Memory Objects or Memory Mapped Files	
POSIX_IO	Truncate_File

Table B-3: Packages and Subprograms under each POSIX.5c Option (Continued)

Package	Subprogram
Sockets Detailed Network Interface	
`POSIX_Sockets`	All except the subprograms below
Sockets Detailed Network Interface and Network Management	

`POSIX_Sockets`	`Set_Flags`
	`Get_Flags`
	`Set_Family`
	`Get_Family`
	`Set_Socket_Type`
	`Get_Socket_Type`
	`Set_Protocol_Number`
	`Get_Protocol_Number`
	`Get_Canonical_Name`
	`Get_Socket_Address_Info`
	`Get_Socket_Address_Info`
	`For_Every_Item`

Synchronized I/O	
`POSIX_IO`	`Synchronize_Data`
Synchronized I/O and Memory Mapped Files	
`POSIX_Memory_Mapping`	`Synchronize_Memory`
Timers	
`POSIX_Timers`	All
XTI Detailed Network Interface	
`POSIX_XTI`	All

a. All versions
b. Return type `Signal_Info`

Annex C: Bibliography

(Informative)

This Annex contains lists of related open systems standards and suggested reading on historical implementations and application programming.

C.1 Related Open Systems Standards

{B1} ISO 8859-1: 1987, *Information Processing—8-bit single-byte coded graphic character sets—Part 1: Latin alphabet No. 1.*

{B2} ISO/IEC 10646:..., *Information processing—Multiple octet coded character set.*

{B3} IEEE Std 100-1988, *IEEE Standard Dictionary of Electrical and Electronics Terms*.

{B4} ISO/IEC TR 10000-2:1998 *Information technology -- Framework and taxonomy of International Standardized Profiles -- Part 2: Principles and Taxonomy for OSI Profiles*.

C.2 Other Documents

{B5} *The Single UNIX Specification: The Authorized Guide to Version 3*. The Open Group, March 2002. UK ISBN: 1-85912-277-9. US ISBN 1-931624-13-5.

Alphabetic Topical Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y