A Graphical Interface to an E-mail Filing, Filtering and Management Tool

Adam J Thornburn

Supervisor: Dr. Alistair D N Edwards

March 2001

Abstract

The automated e-mail filing, filtering and management tool designed and implemented by Russell Odom is a powerful and useful tool for managing e-mail without forcing the user to utilise the filtering tools provided by their choice of e-mail client.

Odom's tool allows flexible scripts to be written to manage the way that incoming e-mail is filtered and managed. Use of the tool allows a very flexible and powerful e-mail management system to be implemented without restricting the end-user's choice of e-mail client, or forcing them to re-write the filtering rules whenever they change or reinstall their e-mail client software.

Whilst the scripting language is as simple as it is powerful, it does require a basic level of programming experience that most e-mail users will not possess in today's connected society where an increasingly large percentage of the population use e-mail for day-to-day communications.

This report describes the design and implementation of a Graphical User Interface (GUI) to enable users with little or no programming knowledge to produce scripts for use with Odom's tool.

Contents

Introduction

Background

Odom’s tool is a powerful and client independent way of managing e-mail, filtering **spam** (unwanted messages) and providing many different types of auto-responders to deal with many situations such as generic away messages (i.e. on holiday, extended business trip), or to provide a standard response to certain types of mail.

The tool is ideally suited for implementation on a Unix based mail server, processing messages for each user as they arrive. Users can operate different e-mail clients to each other, upgrade to the latest version or never check their mail, whilst their mail filters are processed seamlessly in the background.

Unfortunately a prerequisite of using the tool is a basic knowledge of programming and programming languages, in order to make use of the scripting language required to define filters.

The user is clearly in the best position to decide which (if any) filters would suit their needs best, and therefore it is important that any user of the e-mail tool be able to specify the filters that they desire, whether they have a previous background in programming languages. Even for a user with a basic understanding of the concepts involved in writing a script for the tool, much a of the power of writing the scripts by hand would be rendered irrelevant by a system that allowed them to specify quickly and simply basic filter rules without having to learn and debug the scripting language themselves.

Aims and objectives

This project aims to generate a practical solution to this problem by implementing a a Graphical User Interface (GUI) which will allow the user to generate a series of filters using a simple, easily understood graphical application which will automatically generate the required script files to implement the rules.

The project will assume that Odom’s tool is going to be installed and configured by experienced technical support staff in such a way as each user on a mail server will have a file in their user-space, which defines the filters that will be applied to the users personal mailbox.

The project will try to identify the way most users approach the task of filtering their e-mail, and implement an interface, which is simple and effective to use with little or no tuition, whilst retaining as much of the original power of the scripting language as possible.

The final implementation should allow the user to specify the same rules as they could when writing the script files by hand, but without a steep learning curve, with most commonly used filters being applied with a few keystrokes, and more complex rules using the same straightforward system.

Assumptions

This project will assume that Odom's tool will be implemented on a Unix based e-mail server in such a way as all mail arriving at the server will be passed through a unique filter specification for each user. This set-up would be implemented by technical support staff and therefore is not relevant to this report. It should be noted however that the pipeline structure of Odom's tool lends itself ideally to this kind of installation.

Each user would create a filter file based on Odom's filter grammar which would specify their individual filter preferences. This method would make the filtering system completely independent of the e-mail client, and would allow the processing of the user's mail without any interaction (i.e. checking mail) on the user's behalf.

It is also assumed that the user's desktop platform will be Microsoft Windows and that the version will be new enough to include the interface first utilised in Windows '95 but largely unchanged in Windows 98 and ME, Windows 2000 and the forthcoming Windows XP.

Based on these assumptions this report will concentrate on the development of a Microsoft Windows based solution to the design and implementation of a Graphical User Interface (GUI) for producing the filter files required by Odom's tool.

Research

Structure of an e-mail

The mechanics and structure of an e-mail are not relevant to this report, except to determine the varieties of filter a user might wish to specify, Odom's report and implementation handles this issue, and it is sufficient within the scope of this report to understand that an e-mail contains two parts, a header and a body. The header consists of numerous fields such as *to, from* and *subject*. The fields that are available to this implementation are described later in this report when we look at the grammar used by Odom's tool. The body of the e-mail is simply the message text, which is usually formatted as plain text, or increasingly in Hyper Text Mark-up Language (HTML), which is the language used to format web pages. The formatting of the body again does not apply to this report as it will not affect our implementation of the GUI.

Uses of filtering software

Understandably other people were reluctant to allow that author free reign of their personal mailing systems to establish the sorts of filters that they were implementing in everyday use. Therefore these suggestions are based largely on my personal mailing habits and experience of mailing systems I have used or configured in business and home environments, in both professional and personal capacities.

Spam: Many people in both business and home environments receive a large number of un-requested mailings on a regular basis. These mails are know as Spam, and a large amount of user's time is wasted determining the value of these mails and then removing them from their inbox. Once a user has become the subject of spam mailings, it is very difficult to avoid continuing to receive the mailings. Unlike mailing lists these e-mails are unsolicited and therefore give you no option to unsubscribe from their lists, which may be passed freely, or sold between one spamming again and another.

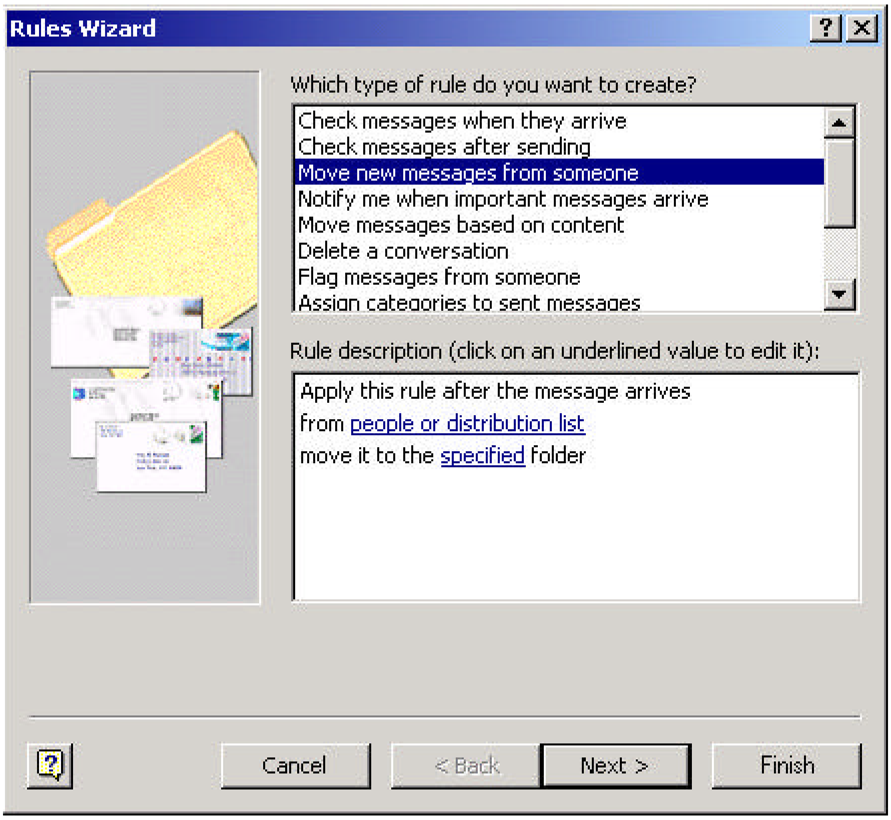
The only way to be certain to remove a user's address from all Spam mailing lists is to give that user a new address. This, however can be very inconvenient, not just spamming agents will be aware of this address, but also all of that user's legitimate contacts will also know and be used to that address. Given that the user cannot change their address, it would be very useful if the e-mail filtering tool could be used to catch and remove spam mailings before the user is forced to sift through all the mails. If the user is concerned that important mails may be deleted the system could instead put mailings it believes to be spam mail to the lowest possible priority so that the user can concentrate on more relevant mails first.

Auto responders: If a user is going to be away from their computer for an extended period of time (i.e. when they are on holiday), or if the user receives many mailings related to a similar subject (i.e. in a technical support role), then it may be useful for a generic message to be returned to the sender until such time as the user can send a more personal response. A filter could be used to catch messages to which this applies and send a predefined message to the sender before filing the message as normal for future attention.

Multiple mailboxes: Many e-mail users, that author included receive many e-mails each day. In order to make these e-mails easier to find and organise it is possible to use more than one named mailbox, and Odom's tool

Outlook 2000 Filters

Outlook 2000 has a powerful filter system built directly in to the program. On the surface the system of filter specification is quite intuitive. There is a long list of possible rules to choose from. Selecting one of these rules gives a textual description of the action this rule will perform with the variable options (such as folder to move to, or the contents of a specific field) underlined. Clicking on an underlined option gives the user the option to define those variables.



Which type of rule do you want to create?

- Check messages when they arrive
- Check messages after sending
- Move new messages from someone
- Notify me when important messages arrive
- Move messages based on content
- Delete a conversation
- Flag messages from someone
- Assign categories to sent messages

Rule description (click on an underlined value to edit it):

Apply this rule after the message arrives
from <u>people or distribution list</u>
move it to the <u>specified</u> folder

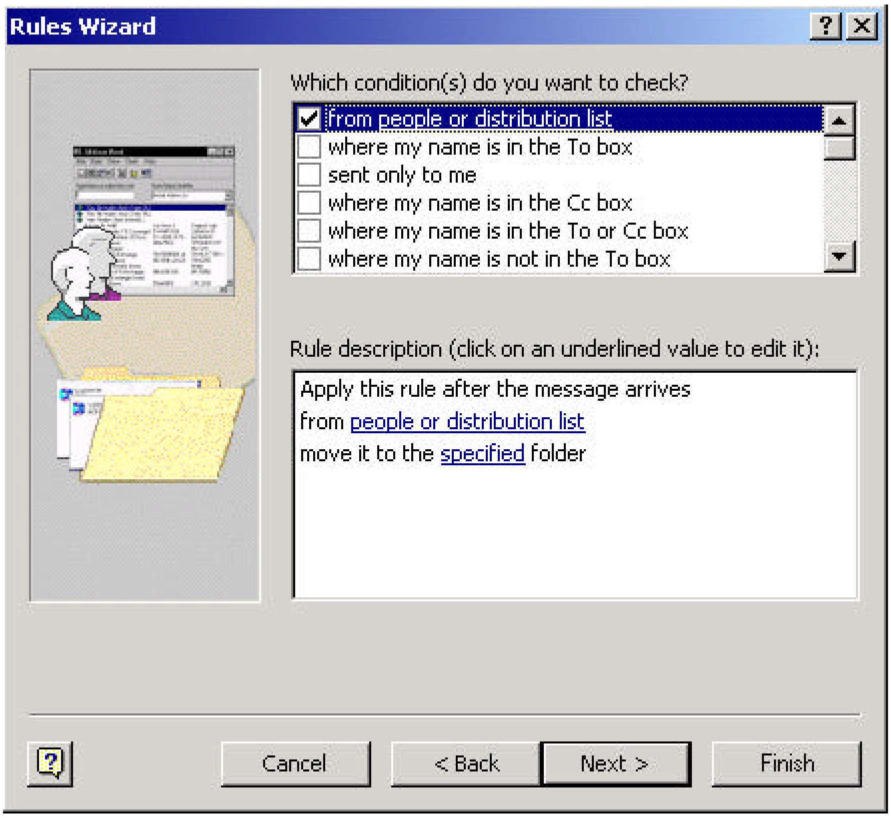
Figure 1 – Outlook 2000 filter specificaton

In practice this system can be quite confusing to use, the large number of initial options can be confusing and may lead to the wrong rule being chosen for a specific task. There are also several ways of creating the same rule from different parts of the application which whilst useful to a more experienced user could be considered less that intuitive by a novice to the system.

The advantages of this system are the use of simple English statements to describe the functionality of a filter. This means that establishing exactly what purpose a filter was created for is a simple process. However the large number of options presented to the user at each stage in order to provide a full compliment of filter types using this interface may cancel out this benefit.

Another possible drawback is that Microsoft has used a specially designed interface component (the textual description with underlined variables) which users may not be familiar with, and may find confusing.

It is worth noting that of the e-mail clients considered in the report, Microsoft's is the only application to offer context sensitive help within its rule creation interface. This may be due to the more complex nature of Microsoft's interface, but the fact the neither Netscape nor Opera offer any sort of assistance outside of the online manual may be an oversight when catering for less advanced users.



Which condition(s) do you want to check?

- ☑ from people or distribution list
- ☐ where my name is in the To box
- ☐ sent only to me
- ☐ where my name is in the Cc box
- ☐ where my name is in the To or Cc box
- ☐ where my name is not in the To box

Rule description (click on an underlined value to edit it):

Apply this rule after the message arrives
from people or distribution list
move it to the specified folder

Figure 2 – Outlook filter specification

Netscape 6 Filters

Netscape's filter system remains largely unchanged from that implemented in version 4 of the program. The filter interface is composed of just two different windows. The first is a list of the individual filters by name. This allows new filters to be created and existing filters to be edited or deleted. It also allows the order in which filters are applied to incoming mail to be altered.

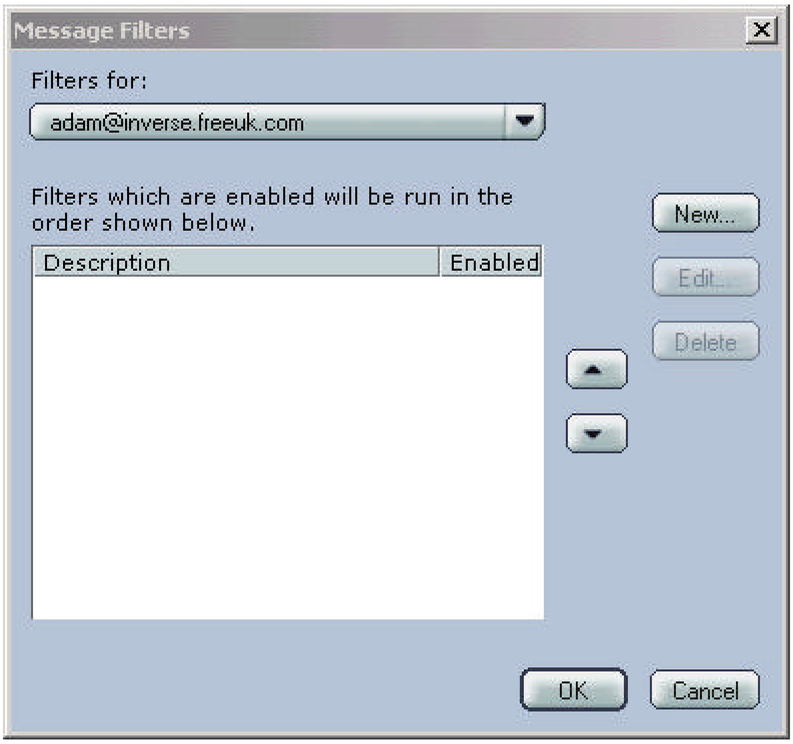


Figure 3 – Netscape 6 filter list

The second window defines the structure of the filter and is straightforward in use. The main option for each filter is whether just one or all of the conditions must be satisfied for the rule to be applied.

Any number of conditions may be defined for each filter, each condition is composed of a field to check, the type of match to be made and the value to compare it with. For example a user might specify the following condition:

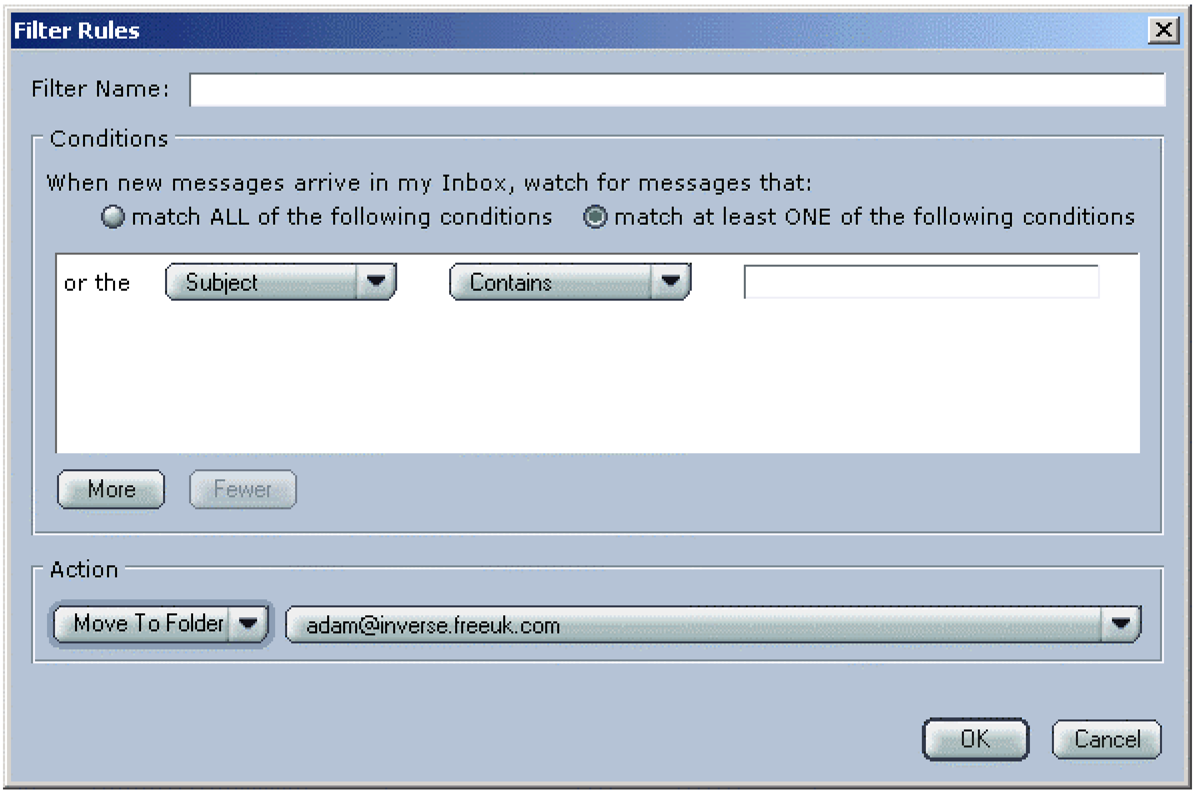
From equals ted@piebald.com

Where the field to check is the 'from' field, the match type is 'equals' and the value to compare with is 'ted@piebald.com'.

Only one action may be specified for each filter, and typically each action has a type and value. The type defines the sort of action to be performed if the conditions are met (i.e. move message to a folder) and the value depends on the action type, so in the case of a move to folder action, the value will specify to which folder the message should be moved.

The process of creating a new filter is straightforward and intuitive, despite Netscape's use of a proprietary interface design in version 6 of their Internet suite.

Selecting 'new' from the main filters window brings up the filter rules dialog box (shown above), which allows the complete specification of a filter from one window. The name of the filter, along with the type, any number of conditions and the action performed if the conditions are met are specified and 'OK' is clicked.



Filter Rules

Filter Name:

Conditions

When new messages arrive in my Inbox, watch for messages that:

○ match ALL of the following conditions ● match at least ONE of the following conditions

or the Subject Contains

Action

Move To Folder adam@inverse.freeuk.com

Figure 4 – Netscape 6 filter specificaton

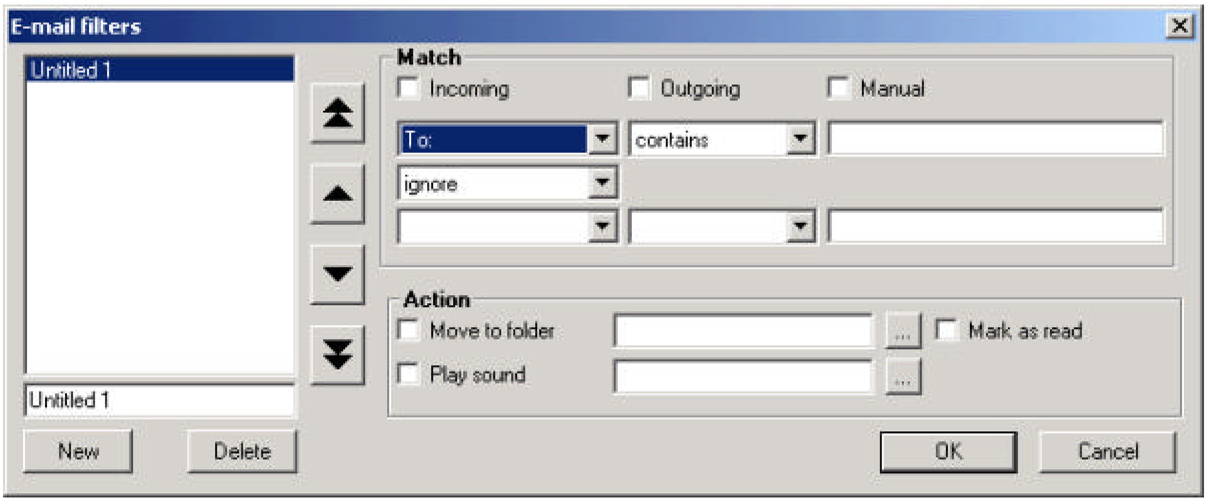
The filter is then added to the list in the main filters dialog. From there this filter can be deleted, edited, or its position within the list of filters changed.

Opera 5 Filters

The mail client within Opera 5 uses a single dialog approach for the interface to its main filter specification. It is the most limited system of all the e-mail clients examined in this report.

Unlike Microsoft Outlook 2000 and Netscape 6, Opera only allows a limited number of conditions to be specified for a each filter. A filter may have only one condition, or it may have two conditions, with the option of whether both or just one condition must be met.

Even more limiting is the fact that there are only two types of action which may be applied for any given filter, to play a sound and to move the message to a specific folder. Either or both of these actions may be specified, but there is no scope for auto responders, or other more complicated and advanced filters.



Untitled 1

Match
☐ Incoming ☐ Outgoing ☐ Manual
To: contains
ignore

Action
☐ Move to folder ☐ Mark as read
☐ Play sound

Figure 5 – Opera 5 filter specification

The interface looks a little cluttered as the information, which is spread across two dialogs under Netscape, is presented in only one in Opera. This does not make the interface considerably more difficult to use, but this is testament to the limited number of options available to the user.

Another drawback of the Opera single dialog interface is that of the cancel button. In the event of an error the cancel button (or escape key) can be used to close that dialog without saving any changes. It is however unclear within this interface if after creating several new filters, whether pressing cancel will delete all of these filters, or just clear any changes to the current filter.

Other e-mail clients

There are many other mainstream e-mail clients available both for Windows and for other operating platforms. The filtering systems implemented in such clients as Eudora, Pegasus, Outlook Express, were either non-existent, or not sufficiently unique to warrant inclusion here.

Clients for platforms other than Microsoft Windows were not considered, as the interface would not be based on Microsoft Windows' standards.

Languages

The language chosen to implement the GUI need not be especially powerful as there will be little complex processing performed by the application. It is more important that the interface can be readily created, altered and tested. It is therefore expected that a Rapid Application Development (RAD) tool will be used for the final implementation.

The following languages/development environments were considered for the implementation of the GUI,

?? *Perl*

?? *C/C++*

?? *Java*

?? *Visual Basic*

?? *HTML/XML/JavaScript*

?? *Delphi*

Despite Odom's tool being based around Perl, it would be entirely inappropriate to attempt to produce a GUI using this tool. Perl is mainly used to manipulate information often in conjunction with web-based forms, and whilst this is one possible solution, it does not allow the RAD envisaged by the author.

Similar difficulties apply to C/C++. Whilst both Borland/Inprise and Microsoft have advanced and powerful Integrated Development Environments (IDEs) available for the C++ language, they still over complicate the process of producing a Graphical User Interface in favour of power and flexibility.

For these reasons both Perl and all C derivatives were not considered for the final implementation, however the other languages were considered in more depth.

Java: This language has the advantage of being a cross-platform; this would allow the final implementation to run on any platform capable of supporting Java applets. The drawbacks however are quite complex. Interfaces for Java under Windows (the target platform) can take two forms.

When using Java within the Microsoft J++ environment, standard Windows controls are available which will work under Windows but not on other platforms, which removes the benefits of a cross-platform languages, but allows a standard interface to be developed.

Another option is to use Swing components, which is a library of GUI components designed for use across operating system platforms. This would allow the development of a cross-platform implementation, but would introduce non-standard components to the implementation, which might make the implementation less intuitive to a basic Windows user, which is the target audience.

Visual Basic: This is a RAD tool, based on the original BASIC language and extended for use under Windows. It is the most widely used programming tool in the world, because of its easy of use and short development cycle.

Whilst Visual Basic executables are not the most efficient possible and rely on several run-time libraries for execution, it has the advantage of almost instant compilation within the IDE and a drag and drop interface design tool that makes producing the interface to an application more like a drawing package than a programming tool.

This is very powerful for designing user interfaces, as all of the standard Windows components are available and can be created, deleted, resized and edited in a simple and most importantly quick manner.

This tool is ideally suited to the project as it will allow easy reconfiguration of the interface as development of the implementation progresses, whilst the inefficiencies of the executables will be rendered irrelevant by the minimal amount of processing actually required by the application.

There is also the advantage that the author is familiar with this tool.

***HTML/XML/JavaScript*:** A solution could be created using a web browser to display the interface, with the actual GUI designed in HTML and Dynamic HTML (DHTML) with the processing and interface control handled with browser scripting language JavaScript.

Advantages of this approach would be the cross-platform nature of the web, and the graphical and less restrictive format of HTML. However design and usability on the web are controversial topics, as it is difficult to find a balance between style and usability. For a project such as this, usability will come from familiarity with the interface concepts, and as such it is probable that a standard Windows interface would be more intuitive than a web based one.

Delphi: This IDE from Borland/Inprise is similar in use to Visual Basic, with the focus on Rapid Application Development, and a similar style interface for creating the interface and specifying properties, methods and events for each component.

Rather then using BASIC as the underlying language, Borland have chosen Object Pascal instead, Borland have also chosen to avoid run-time dependencies and include a high quality native code compiler as part of the package. This makes Delphi executables, smaller and more efficient than Visual Basic equivalents.

User Interface design

The most important aspect of the design process is the graphical interface presented to the user. This interface must be as powerful as possible whilst retaining an intuitive method of working that will be obvious to any user familiar with the Microsoft Windows way of working.

Having considered the different approaches to the problem of a filtering interface chosen by other developers, it is clear that there are many different methods, some more intuitive than others. It can also be observed that each interface utilises different components to present the options and information available. Some use standard windows components, others, including Microsoft's offering, utilise a proprietary method.

When designing an interface there are some (often flaunted) guidelines available from Microsoft specifying general rules for the layout of a Windows dialog. These rules are only a rough guide and measurements are approximate, but these rules if followed should result in a clear interface readily accessible due to its familiar appearance and functionality.

By following standard design protocols less explanation of the function of each control will be necessary, resulting in an uncluttered and functional interface.

Microsoft MSDN User Interface Deign Guidelines

Included below are relevant extracts from the 'Windows User Experience' book on the Microsoft website. There is considerably more content on the site than would be appropriate to include here, for further reading on this subject consult the Microsoft website at the address(es) show in the references section.

Measurements in this section are given in Dialog Units (DLUs). One horizontal dialog unit is equal to one-fourth of the average character width for the current system font. One vertical dialog unit is equal to one-eighth of an average character height for the current system font. The default height for most single-line controls is 14 DLUs

Size of Common Dialog Box Controls

Control	Height (DLUs)	Width (DLUs)
Dialog boxes and property sheets	263 max. (for 640 x 480 screen resolution) 218 215 188	263 max. (for 640 x 480 screen resolution) 252 227 212
(For property sheets, heights include 25 DLUs for property sheet button bars.)		
Command buttons	14	50
Check boxes	10	As wide as needed
Drop-down combo box and drop-down list	10	Size to match other drop-down combo boxes and text boxes
Option buttons	10	As wide as needed
Text boxes	14	Size to match other drop-down combo boxes and text boxes
Text labels	8 per line of text	As wide as needed
Other screen text	8 per line of text	As wide as needed

The diagrams below show the standard spacing and alignment of controls in a dialog, once again, these are not strict rules but indicate a general method of alignment and positioning which should be adhered to.

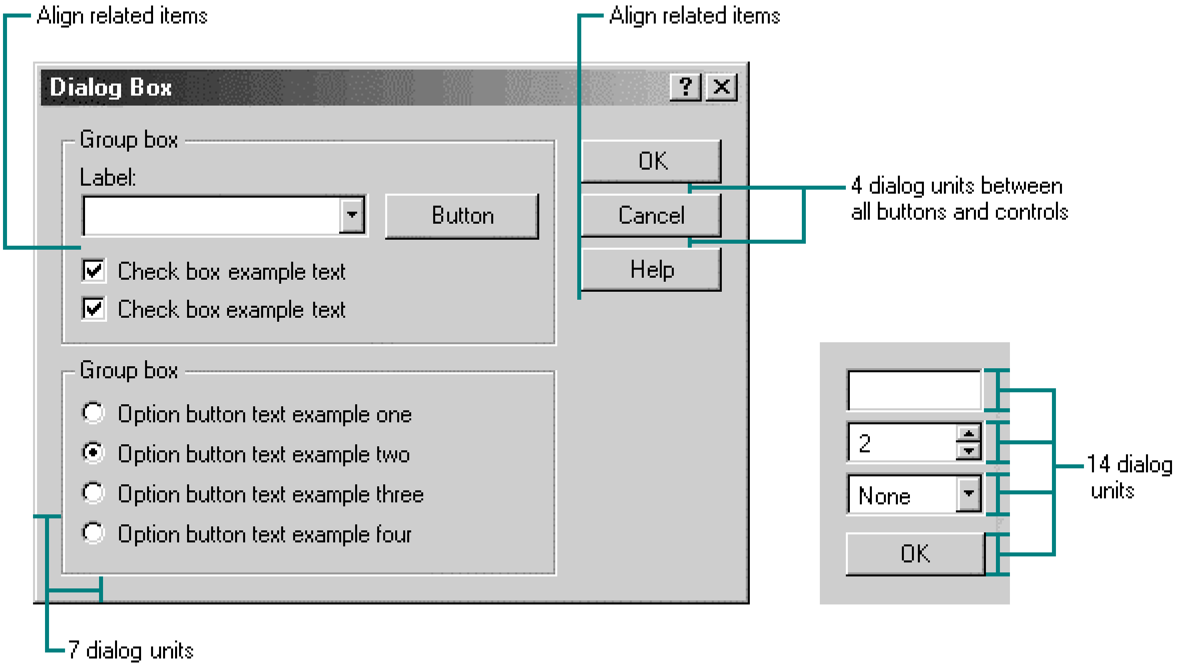
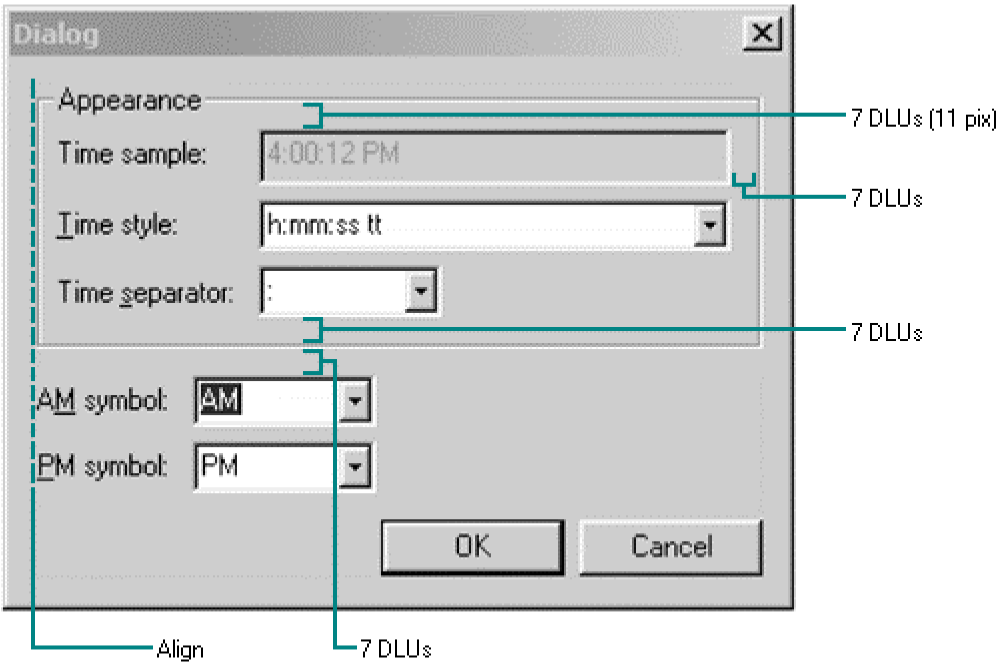

Align related items
Align related items
Dialog Box
?
Group box
Label:
Button
OK
Cancel
Help
4 dialog units between all buttons and controls
☑ Check box example text
☑ Check box example text
Group box
○ Option button text example one
● Option button text example two
○ Option button text example three
○ Option button text example four
2
None
OK
14 dialog units
7 dialog units

Figure 6 – Microsoft design guidelines



Time sample: 4:00:12 PM
Time style: h:mm:ss tt
Time separator: :
AM symbol: AM
PM symbol: PM

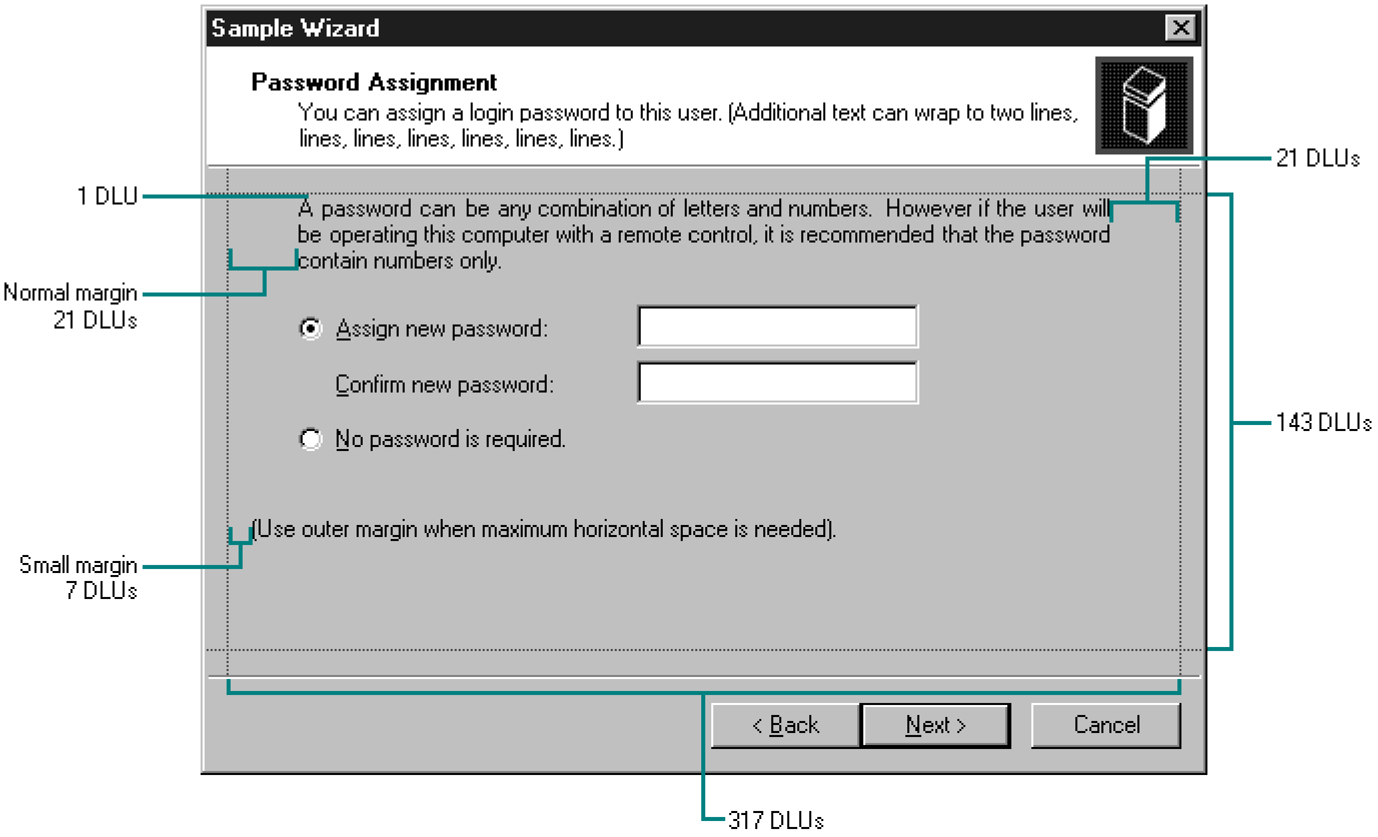
Figure 7

Spacing Between Interface Items

Interface items	Use this spacing (DLUs)
Dialog box margins	7 on all sides
Between paragraphs of text	7
Between text labels and their associated controls (for example, text boxes and list boxes)	3
Between related controls	4
Between unrelated controls	7
First control in a group box	11 down from the top of the group box; align vertically to the group box title
Between controls in a group box	4; align vertically to the group box title
Between horizontally or vertically arranged buttons	4; align vertically to the group box title
From the left edge of a group box	9; if the group box is left-aligned, controls are 16 from the left edge of the dialog box or property page
Last control in a group box	7 above the bottom of the group box
Smallest space between controls	2
Text label beside a button	3 down from the top of the button
Check box, list box, or option button beside a button	2 down from the top of the button

Wizards

Microsoft also provides style guides for wizards, which may be useful given the nature of the project. A wizard is a step-by-step tool typically for simplifying some operation by breaking it up into several steps each in its own dialog. The diagram below shows the standard layout for a wizard dialog.



Sample Wizard

Password Assignment

You can assign a login password to this user. (Additional text can wrap to two lines, lines, lines, lines, lines, lines, lines.)

A password can be any combination of letters and numbers. However if the user will be operating this computer with a remote control, it is recommended that the password contain numbers only.

● Assign new password:

Confirm new password:

○ No password is required.

(Use outer margin when maximum horizontal space is needed).

Figure 8 – Microsoft wizard design guidelines

Wizard dialogs are an especially useful tool for small applications where an intuitive interface is essential. There is an area at the top of each dialog for a short description of what function the controls on the dialog perform, dispensing with the need for complex and separate help files or manuals. Their interface is also consistent with certain buttons expected to appear on each dialog for easy navigation through the steps.

Microsoft's extensive use of wizards in its newer operating systems (Windows 98, ME, 2000) and software, mean that most Windows users are familiar with this method of configuration

User's familiarity with the now standard wizard format has led other developers to design their products around this standard interface. Notably InstallSHEILD the industry leading tool for producing custom installation routines for software now outputs installer programs with a wizard driven interface.

With such a proliferation of small applications based on a standard interface such as this it is safe to assume that most if not all Windows users will have encountered this style of interface in the past, and will have a reasonable expectation of how it will function.

Design

Chosen language

Of all the languages discussed in the previous section, the two most appropriate for this implementation are Visual Basic and Delphi. They are both Rapid Application Development tools, in both cases with emphasis on easy interface development. In both cases the interface is ‘drawn’ from a standard set of Windows components.

This means that both tools are ideal for the creation of simple Windows applications, with intuitive interfaces, which will use methods familiar to Windows users.

Delphi has the advantage of more efficient compilation, and no run-time dependencies. However the author has previous knowledge of Visual Basic, and the tool is more readily available.

With these factors in mind Visual Basic was chosen as the tool for development. Visual Basic allows fast interface development using the standard windows components, with the underlying BASIC language being simple to use, yet powerful enough for the purpose.

Interface approach

The general approach to the design of the interface needs to be one that is familiar to the end user. There are any number of way of laying out the same controls on a dialog, many of which may be quite intuitive and straight forward to use. However for the most effective interface a design which is already familiar to Windows users would be the best solution. Some example of this are outlined below.

Tabbed dialog: Many preferences and configuration menus use this method for presenting a large number of options to the user. Controls are grouped by function using horizontal rules or group boxes.

Advantages: Many options can be placed on a single dialog with multiple tabs, allowing a central point from which to work, with no need to switch from one dialog to another repeatedly.

Disadvantages: This method can lead to a cluttered interface which may confuse the user.

Standard Windows wizard: This style of interface as discussed in the previous section is common across many of Microsoft and other developer’s software. It is a convenient way of presenting a step-by-step process to the end user whilst retaining an uncluttered interface.

Advantages: Having each step on a separate dialog creates an uncluttered interface. This allows fewer controls per dialog, with all the controls relating to one function of the application. In the scope of this project a separate dialog might be specified for: new filters, creating conditions, creating actions and final output.

Disadvantages: The user is forced to switch between dialogs to make changes to different aspects of the filter configuration.

Two-dialog approach: This is similar to the Netscape filters interface. One dialog presents a list of existing filters, with options for new filters, deleting filters, editing filters and changing the order of filters. The other dialog is an all encompassing filter creation window. It allows the specification of filter information (i.e. name, description), conditions and actions all from the same dialog.

Advantages: Few dialogs, therefore less confusing window switching, creation of a filter is handled by only one dialog.

Disadvantages: When many conditions or actions are specified the filter creation dialog may become very cluttered and complex to use.

Chosen approach: The chosen solution is based mainly on the standard Windows wizard, as its familiar and uncluttered interface should allow for the most intuitive and powerful design.

Attributes of other possible designs may be included; in particular a tabbed dialog approach will be considered and contrasted in more detail.

Condition specification

With the overall design confirmed, it is important that the interface is as powerful as possible whilst retaining an intuitive flexible interface. Odom stated in his own report that the disadvantages of Netscape 4's filter interface included weak specification of conditions, and the imposition of only a single action per filter. In Netscape 4 all conditions in a filer must be satisfied for the action to be applied.

Netscape 6 improved upon the condition specification by allowing a choice of allowing only *one* condition to be met or requiring all conditions to be met for each filter. Clearly this is still considerably less flexible that Odom's filter language (see appendix A) which allows conditions to be specified with any combination of 'and' and 'or' clauses.

Attempting to create a user interface which duplicates Odom's rules for the specification of conditions is clearly unfeasible, at best any interface created would be extremely complex and unwieldy in use, and most likely would require the user to define their conditions in syntax similar to Odom's own implementation.

Some power and versatility must therefore be lost from the filter specification in order to create an effective interface, which is significantly easier to use than Odom's language. A system similar to that employed by Netscape 6 was chosen, which would allow the specification of any number of conditions as defined by Odom but only permitting two permutations of condition checking. Either all conditions but be met or only one condition must be met in order for the filter to be applied.

This reduces the complexity of the interface at the expense of some power, but it can be assumed that any user requiring more power than would be made available would be sufficiently experienced to directly employ Odom's filter language or that in exceptional circumstances technical support staff would assist that user with their difficulty.

Aside from these changes conditions will be applied as Odom's specification, with all match types and fields available to the user.

To further reduce the complexity of the interface the functionality to choose a file for the match values in a condition will be removed. This functionality allows a user to specify a file containing, for example, a list of addresses to match against. Allowing this within the scope of a graphical interface would make condition specification more complex rather than less, as the user would need to create a separate external file rather than performing all functions within the interface itself. Again it is reasonable to do this as the reduced functionality would rarely affect a less experienced user for whom this interface is being developed.

Action specification

Based on the assumption that the interface will be aimed at less experienced users working in a Windows based environment there are several actions in Odom's specification which are not appropriate for inclusion here. These are the '*pipe_to'* and '*system'* actions, which could have unexpected and serious side effect if misused or used incorrectly. Therefore these options will not be made available from the interface.

Unlike the Netscape implementation, which has been the basis for the condition specification, the implementation will allow for any number of actions as specified by Odom. This will allow much more power than is available in the Netscape implementation, as complex actions may be specified such as filing a message for later attention whilst sending an automated reply to alert the sender that their message has been received and will be processed in due course.

Revised language specification

Below is the original language specification devised by Odom, revised with the changes outlined above. This is the language that the final implementation of the user interface will be based upon.

filters ::= filter **|** filter filters
| filter_head filter_spec
filter_head ::= **[filter** name**]**
| [filter name **disabled]**
filter_spec ::= **if** condition **then** actions **end**
name ::= string
condition ::= **expr**
| NOT condition
| (condition **AND** condition**)**
| (condition **OR** condition**)**
actions ::= action | action actions
expr ::= field match_type words **| true | false**
action ::= **file_message** filename
| reply when filename address
| forward_message filename address
| exit
field ::= **from | to | cc | reply-to | subject**
| any_header | body | any | content | domain | sender
match_type ::= **contains | equals | starts | ends**
words ::= word **|** word words

word ::= “string”
when ::= **once | always**
address ::= **sender | postmaster |** sender@domain

Data structure

This filters must be represented internally whilst the user is creating working. When the user has completed the set of filters, this data structure will be parsed and output as plain text following the specification above. This data structure should be as close to the final language output as possible whilst still retaining the ability to be easily modified and extended under execution.

Visual basic allows for user defined types to be created, and as a filter file can be broken into smaller subsections, types may be defined to internally represent those sections.

A filter file contains filters. Each filter contains both conditions and actions. The simplest way to represent this is to create types for conditions and actions with respective properties and make these types subtypes of a filter type.

The properties of a filter are:

- Name
- Enabled/disabled
- Description (not part of the specification, but may be useful)
- Type (all conditions must be satisfied – AND, one or more conditions must be satisfied – OR)

There will be a list of conditions for each filter; each of these conditions will have the following properties:

- Field
- Match type
- Value (word list)

There will also be a list of actions for each filter, these actions will each have these properties:

- Action type
- Filename
- Address

Using these properties it is a trivial task to create a simple data structure in Visual Basic which can store the contents of a set of filters in an hierarchical array of user types which can be expanded as the user adds new filters, conditions or actions, and edited at will until the user is satisfied with the filters they have defined. At this point a simple function will traverse the array and output a text file following the specifications above.

Interface

The chosen interface style is a standard Windows wizard because of the clear step-by-step nature of the task. The basic steps are filter specification, condition specification, action specification and final processing, and the diagram below shows the flow between these states.

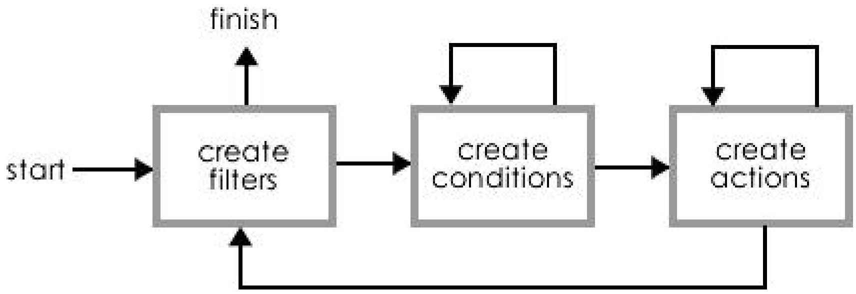


Figure 9 – Flow diagram for the stages of filter creation

It can be seen from this diagram that 4 dialogs are required; create filters, create conditions, create actions and final processing. A separate dialog for the start state is not required, the application could begin at the create filters stage; however it may be useful to include an introductory dialog with a brief explanation of the process to avoid cluttering the other dialogs with unnecessary instructions.

Dialog design

Create filters:

This is the first functional dialog the user will encounter, as with all the other dialogs the user will encounter it will have a white area at the top for a set of brief instruction describing the dialogs purpose and how it should be used.

Also required are a list of existing filters, buttons for adding, deleting and editing filters and a text field to enter a name for new filters.

At the bottom of the dialog, there will be finish and exit buttons. Exit will end execution of the program without saving any work done, fishing will move the user to the final process dialog, where the filter script will be created. The user must create at least one filter before this button will operate.

Create conditions

Aside from the standard instructions and exit buttons, there will a next button which will move the user to the actions dialog, a cancel button which will cancel the creation of the current filter and a list showing the existing conditions for this filter. The next button should not operate until at least one condition has been created.

Two radio buttons will allow the selection of the filter type. For the actual specification of each condition there will be a control for the field, match type and match value. As the field and match type have a pre-defined number of possible values they will be implemented as dropdown lists, match value will be a simple text box where values are separated with a space.

Create actions

This dialog will be very similar to the conditions dialog. There will be next and cancel buttons, a list of existing actions, add, edit, and delete buttons. For the specification of each filter there must be a dropdown list of possible actions, some actions will require an address field and some actions will require a filename field. Both of these will be standard text fields. Only the relevant fields will be made available depending on the action type selected. The filename field should also have a browse button which will allow selection of the relevant file using a Microsoft common dialog for all actions except 'file_message' where the filename refers to a Unix mailbox, and this should be specified as a simple string (i.e. inbox, personal, university, etc.).

Final processing

This dialog needs just one button to output the file to a predefined location for use by Odom's tool. It is envisaged that a configuration file created by technical support staff would specify the location of this file. For the purpose of this report output will be directed to a multi-line textbox on this dialog to allow for easy analysis of the output.

Implementation

Language

The program was implemented in Microsoft Visual Basic Version 6. This is the most up-to-date version available at this time, and includes support for all standard commonly used Windows controls.

Program structure

The data structure and parsing function were included in a separate module called parser.bas. Each dialog is of identical size and initially appears in the centre of the screen to ensure that the user's focus is drawn to the currently active dialog. Non-active dialogs are disabled or hidden to prevent accidental editing out of sequence.

The program begins by displaying the start dialog. This shows a brief overview of the application, and initialises the data structure. The dialogs are moved through following the flow described in figure 9.

Parser/data structure

Based on the properties defined for the data structure earlier the following types and arrays were defined:

```
Public Type tCondition
    type As String
    field As String
    value As String
End Type

Public Type tAction
    action As String
    filename As String
    address As String
End Type

Public Type tFilters
    name As String
    type As String
    description as string
    enabled as boolean
    conditions() As tCondition
    actions() As tAction
End Type

Public theFilters() As tFilters
```

This creates a public array theFilters, which is accessible from all dialogs. The array is of type tFilters, which has the properties name, type, description and enabled. It also contains two further arrays, conditions and actions. This allows each filter in theFilters to contain any number of conditions and actions.

The conditions array in each filter is of type tCondition, which has the properties: type, field and value, whilst the actions array has the properties: action, filename and address.

The parsing function parseFilters traverses the data structure and returns a string containing the scripting language as specified by Odom.

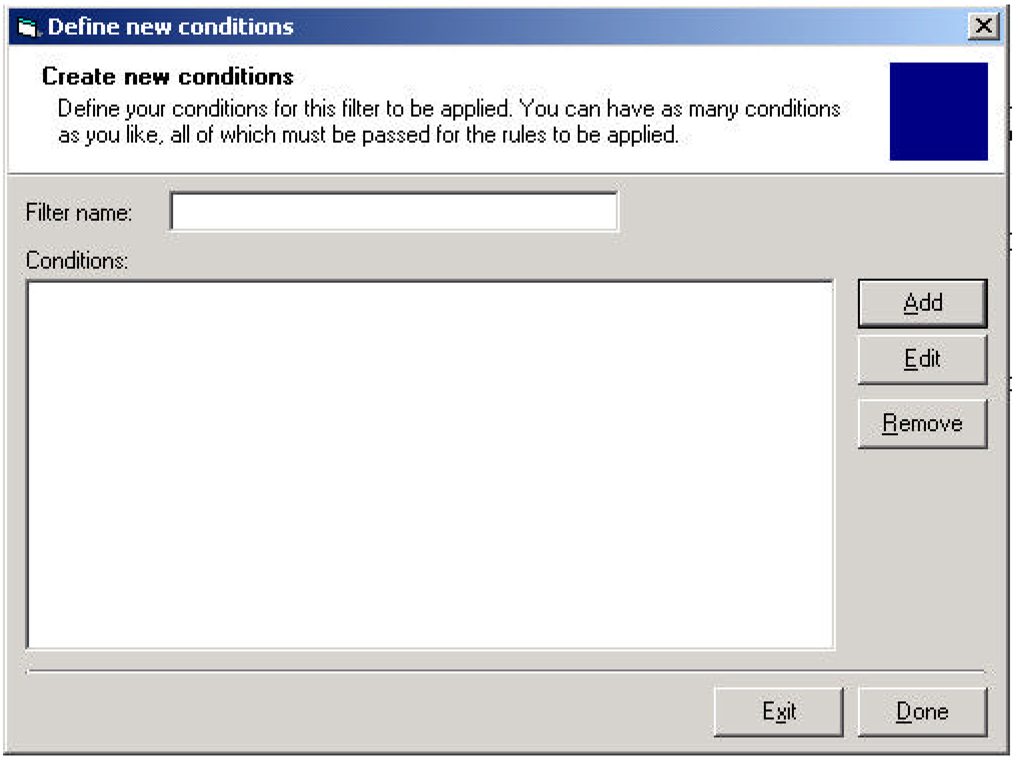
Dialogs

Each dialog contains the controls described in the design section of this report. The conditions and actions dialogs are designed with almost identical layouts to make the interface as easy to learn as possible for the user.

There were three stages of implementation, after each stage a basic user test was conducted along with analysis by the author. After each phase the interface was updated to reflect the discoveries about the accessibility of the interface.

Phase one: Initially a rough interface was developed with many more dialogs than previously discussed. The conditions and actions stages were split over two dialogs each with one dialog to show a list of existing conditions/actions, and another for the actual specification of the rules. This was a fully working verson although the interface was severely limited.

The images below show the initial design of the condition specification dialogs.

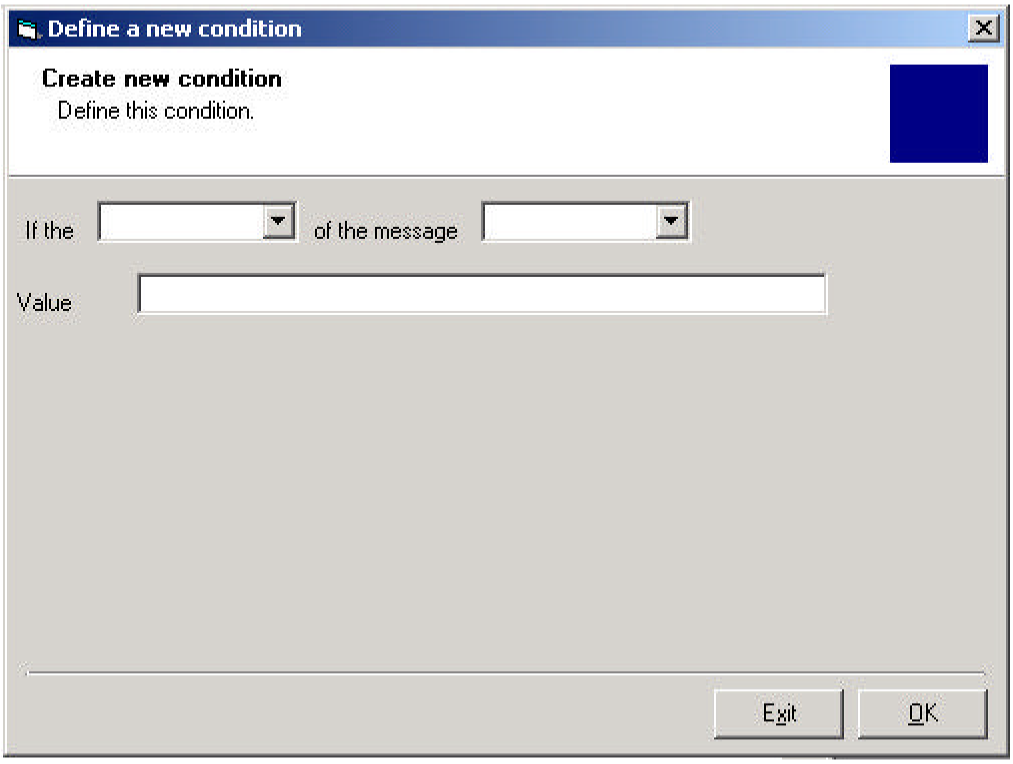


Define new conditions

Create new conditions

Define your conditions for this filter to be applied. You can have as many conditions as you like, all of which must be passed for the rules to be applied.

Figure 10 - Initial design of condition specification dialogs



Define a new condition

Create new condition

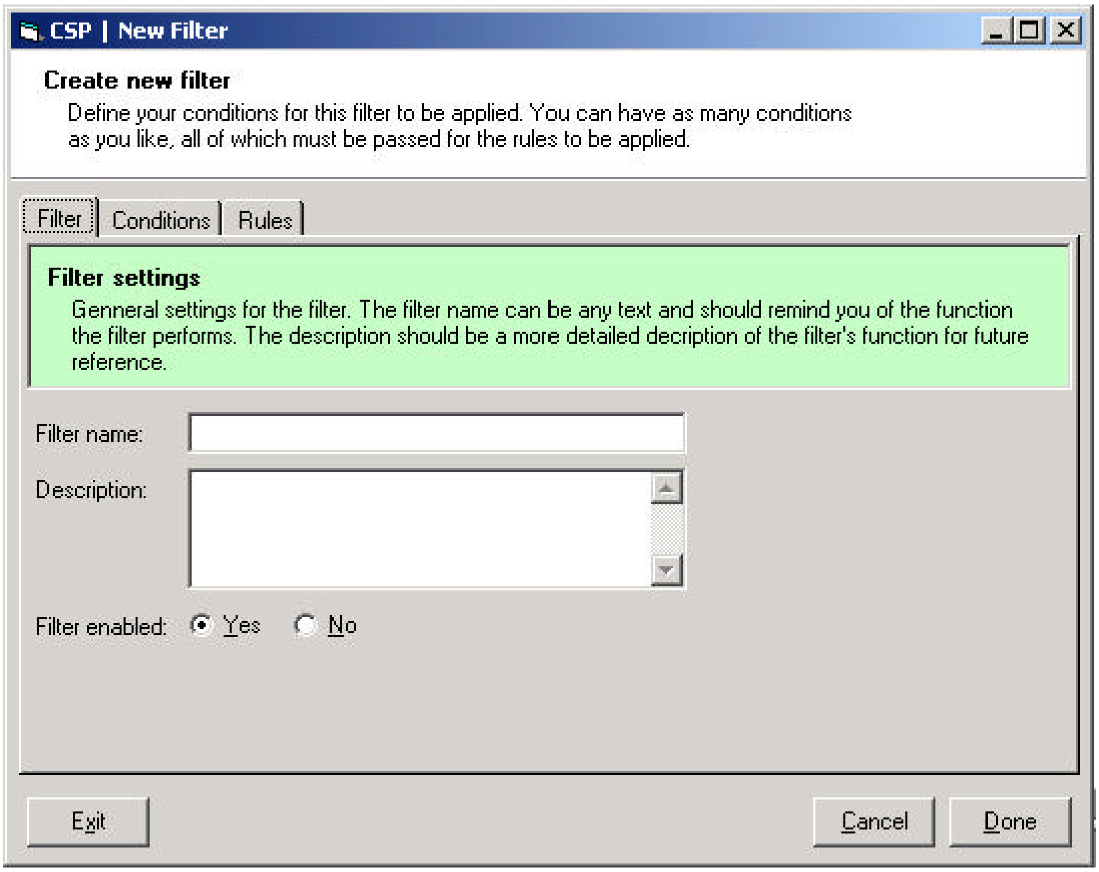
Define this condition.

Figure 11

Phase two: This implementation was an experiment with an interface employing a tabbed dialog approach. This phase was never fully completed as after the initial design was completed it was decided that too much information was included on one dialog.

With filter, condition and action specification all on one dialog it was envisaged that the user would find moving between the different stages more free flowing and less restrictive, however the amount of options and information made for a cluttered and unwieldy interface which strayed too far for the guidelines for wizard design to be intuitive.

The images below show the main dialog with each of the three tabs selected.



Create new filter

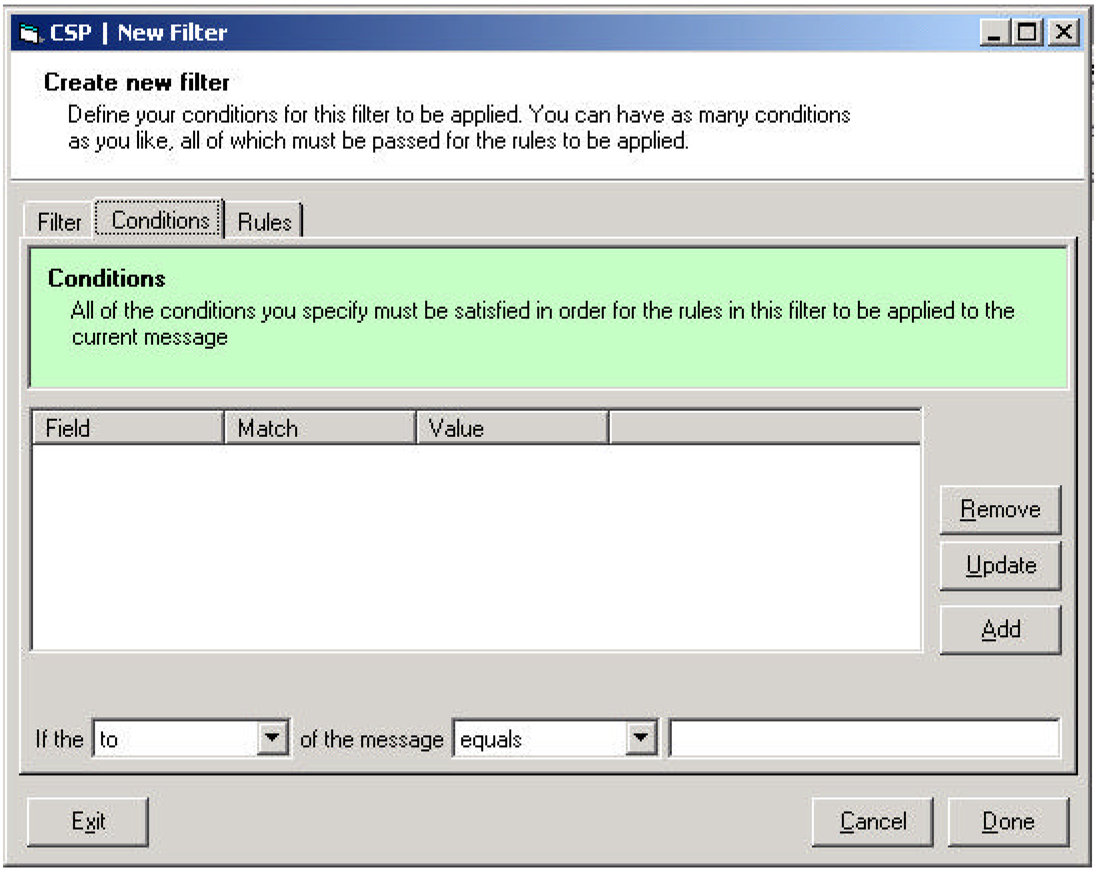
Define your conditions for this filter to be applied. You can have as many conditions as you like, all of which must be passed for the rules to be applied.

Filter settings

Genneral settings for the filter. The filter name can be any text and should remind you of the function the filter performs. The description should be a more detailed decription of the filter's function for future reference.

Filter enabled: ● Yes ○ No

Figure 12 - Tabbed dialog approach filter specification



Create new filter

Define your conditions for this filter to be applied. You can have as many conditions as you like, all of which must be passed for the rules to be applied.

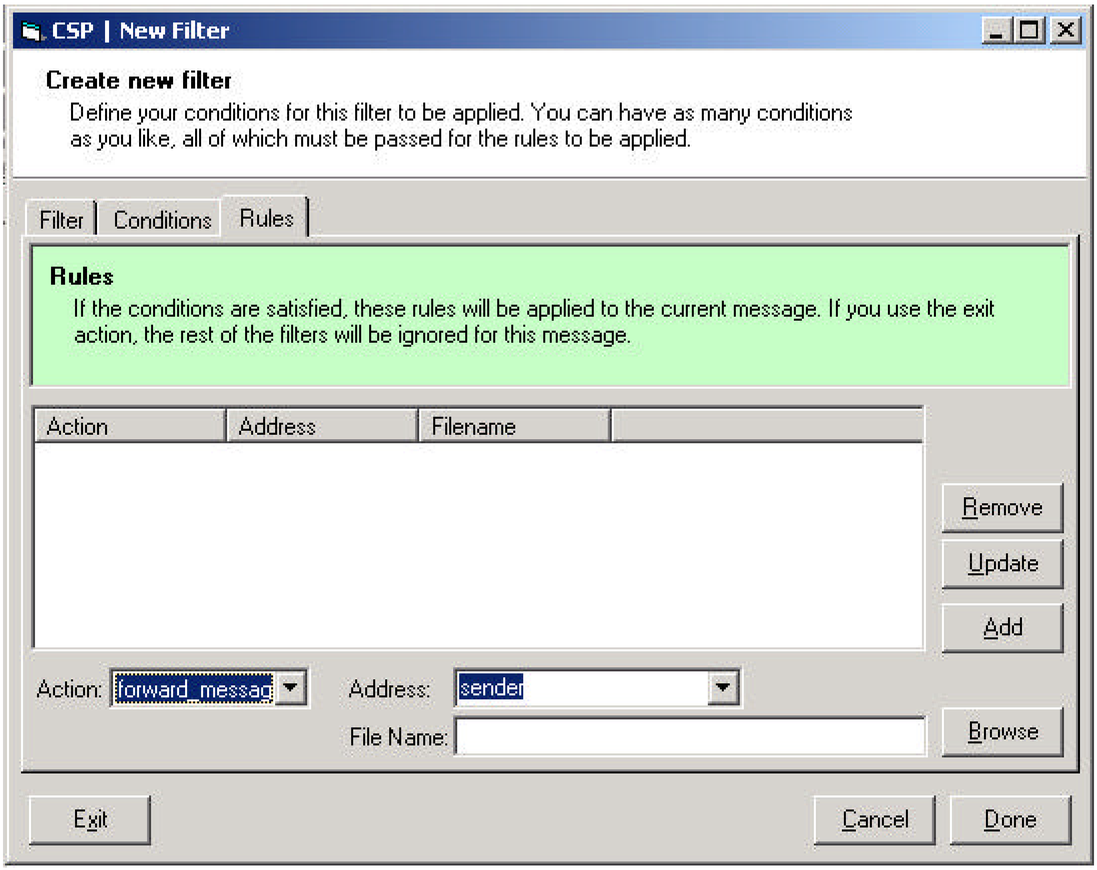
Conditions

All of the conditions you specify must be satisfied in order for the rules in this filter to be applied to the current message

Field	Match	Value

If the to of the message equals

Figure 13 - Tabbed dialog approach with condition specification



Create new filter

Define your conditions for this filter to be applied. You can have as many conditions as you like, all of which must be passed for the rules to be applied.

Rules

If the conditions are satisfied, these rules will be applied to the current message. If you use the exit action, the rest of the filters will be ignored for this message.

Action: forward_messag

Address: sender

File Name:

Figure 14 - tabbed dialog approach with action specification

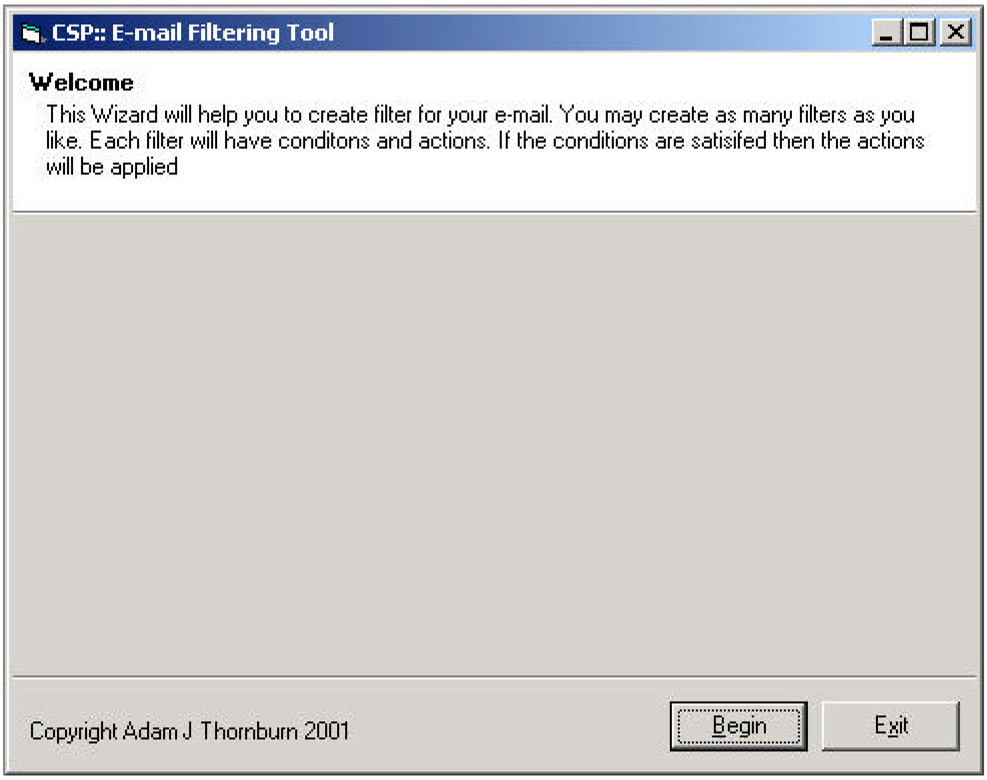
Phase three: This is the final implementation, and builds upon the successes and failures of the previous two implementations. The interface is based as planned on phase one, but considerably overhauled.

Consolidating the contents into three main dialogs has reduced any confusion caused by the large number of dialogs, present in the initial implementation. There are now filters, conditions and actions dialogs as specified in the design section.

These dialogs now include both the existing filters/conditions/actions as well as the controls for creating new instances. As well as reducing the number of dialogs this helps prevent duplication, as the user can see what rules they have already designed when specifying new ones.

The controls are grouped in frames on the dialogs, this is not essential as each dialog only contains controls relevant to one section of the wizard, but they help draw the users eye to the most important aspects of the dialog.

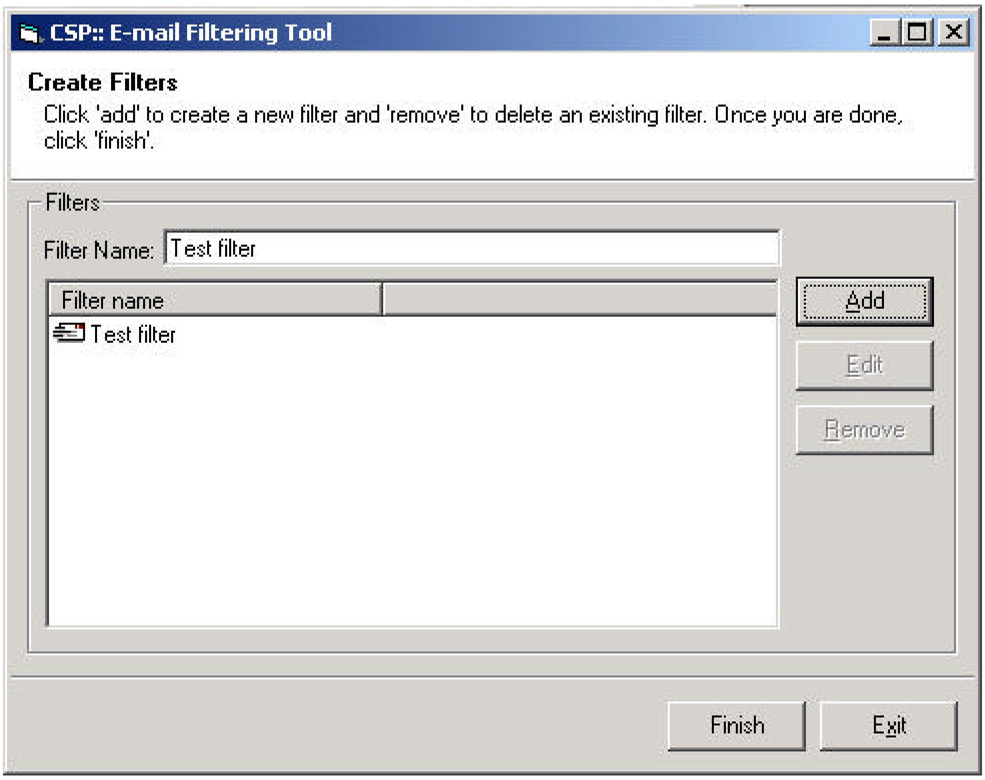
The images below show all of the dialogs available in the final implementation.



Welcome

This Wizard will help you to create filter for your e-mail. You may create as many filters as you like. Each filter will have conditons and actions. If the conditions are satisifed then the actions will be applied

Copyright Adam J Thornburn 2001



CSP:: E-mail Filtering Tool

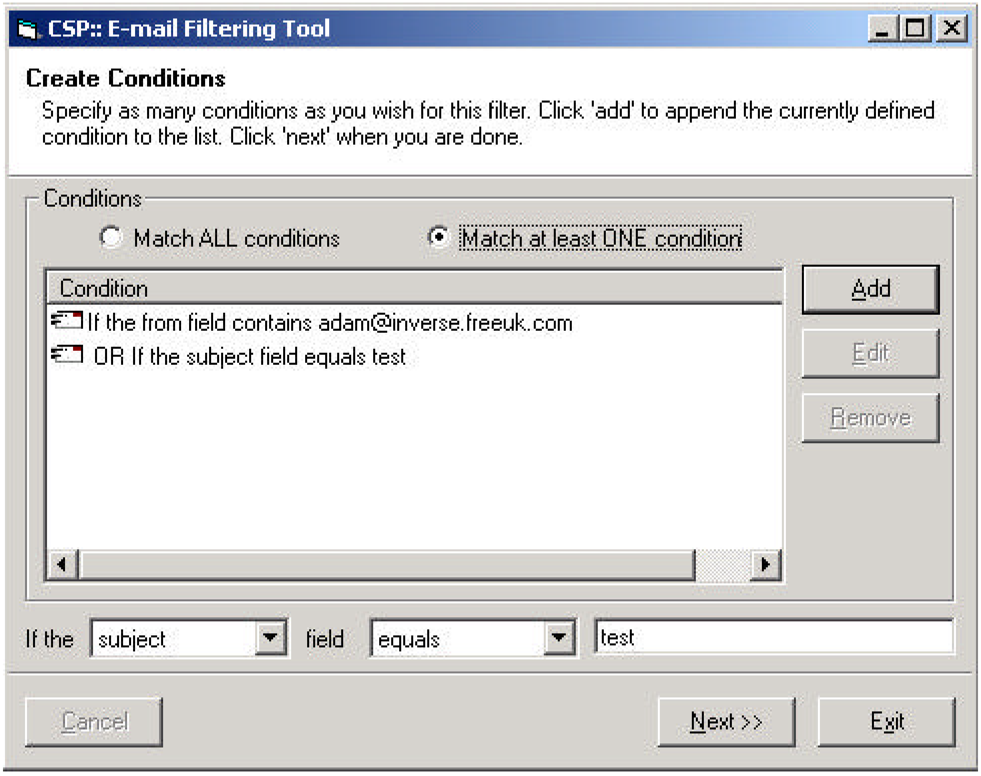
Create Filters

Click 'add' to create a new filter and 'remove' to delete an existing filter. Once you are done, click 'finish'.

Filters

Filter Name: Test filter

Filter name	
Test filter	



CSP:: E-mail Filtering Tool

Create Conditions

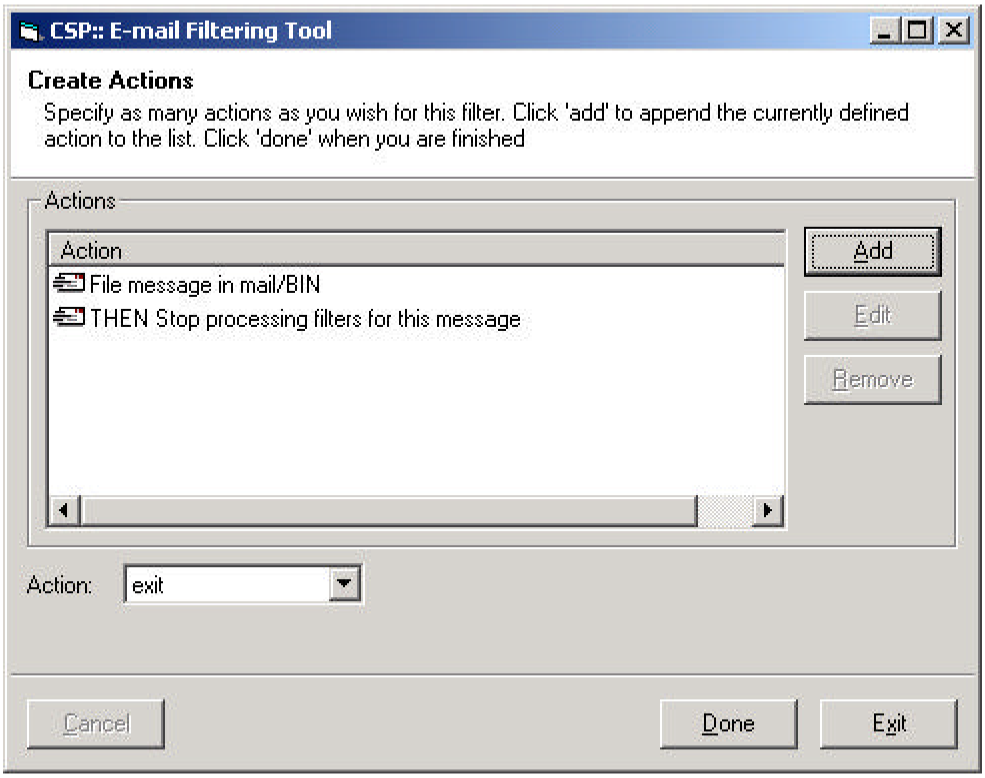
Specify as many conditions as you wish for this filter. Click 'add' to append the currently defined condition to the list. Click 'next' when you are done.

Conditions

○ Match ALL conditions ● Match at least ONE condition

Condition
If the from field contains adam@inverse.freeuk.com
OR If the subject field equals test

If the subject field equals test

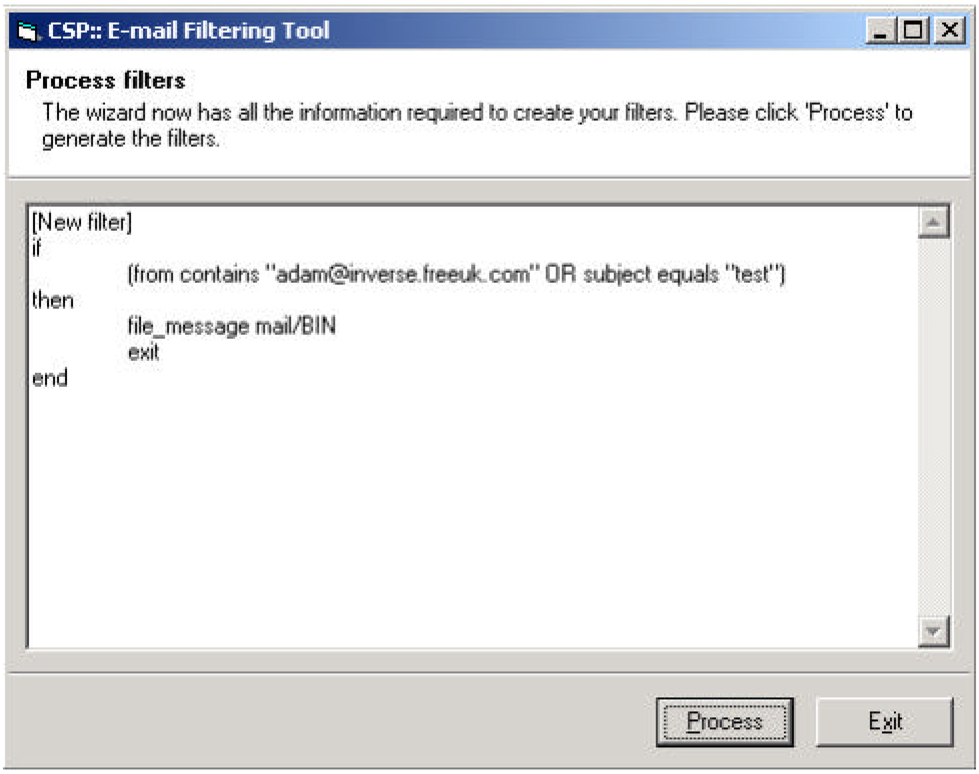


Create Actions

Specify as many actions as you wish for this filter. Click 'add' to append the currently defined action to the list. Click 'done' when you are finished

- File message in mail/BIN
- THEN Stop processing filters for this message

Action: exit



Process filters

The wizard now has all the information required to create your filters. Please click 'Process' to generate the filters.

```
[New filter]
if
        (from contains "adam@inverse.freeuk.com" OR subject equals "test")
then
        file_message mail/BIN
        exit
end
```

Testing

Platform

The tests were run on a Pentium II 450 based PC with 128MB RAM, running Microsoft Windows 2000. The edition Visual Basic used for the final implementation was version 6. Where the application would normally have output the scripts to a plain text file for use by Odom's mail filtering tool, the results were instead displayed in a multi-line textbox on the final dialog. This was to make the out put easier to check as the interface was never actually used with Odom's tool, due to the complexity of setting up and using a Unix mail server. Instead the output was checked to ensure it was syntactically valid.

Syntax checking

In order to ensure that the output generated by the application was valid for use with Odom's tool, numerous fictitious filters were created and the output analysed by hand to ensure that it followed the specifications set out by Odom, and included in this report in appendix A. It was for this reason that the parsing function 'parseFilters' output the correct amount of white space, as although Odom's tool ignores white spaces in the filter text it makes it much easier for a human to check the output by hand.

User testing

Several inexperienced computer users, who used e-mail on a Windows based platform on a regular basis were asked to create filters using the tool, with no assistance from the author.

The users started the task with the tool loaded and displaying the initial dialog. No other applications were loaded, and users were asked only to use the program running.

Tasks

For testing each user was asked to create a filter file containing three filters.

Filter 1: Create a filter which will file any messages from adam@inverse.freeuk.com or ajt111@cs.york.ac.uk in a file called 'self'

Filter 2: Create a filter which will take any messages that contain the word 'bonus' in any field or '$$$' in the subject field and file them in a file called 'spam'

Filter 3: Create a filter which will return the message contained in the file 'faq.txt' to the sender of messages addressed to 'faq@helpme.com'

Results

The users were first asked to create these filters on the initial implementation, which whilst fully functional was missing some vital parts of the interface. None of the users tested were able to complete the tasks, as was expected, but none the less their feedback was useful in confirming the authors intended final interface.

The users confirmed that the large number of dialogs was confusing at that it would be useful to be able to see a list of filters/conditions/actions at the same time as creating new ones.

They also commented on the inconsistent layout and the fact that the interface looked generally untidy. This was to expected from the initial implementation. One omission that was particularly pertinent was when attempting filter 3 the users were all uncertain how to return a message to the sender, as it was not made clear the the keyword required in the address field was ‘sender.’

To overcome this problem in the final implementation, the address textbox on the actions dialog was replaced with a dropdown list, with ‘sender’ and ‘postmaster’ keywords. This allowed the users to type an address of their choice, or slect from one of the two possible keyword addresses implemented by Odom.

No tests were run on the tabbed dialog implementation as the author felt the interface to be too confusing, and a working version was never developed.

The tests were repeated for the final implementation, and whilst largely successful, led to several minor refinements. Some difficulties were raised due to the incomplete implementation, which meant that if a mistake was made and an action confirmed there was no way of removing the erroneous entry. This is not a flaw with the interface as such, but an issue of implementation with will be discussed later.

Several users raised the issue of the way the lists of conditions and actions were displayed. This was in a three column format as seen in the tabbed dialog implementation, which simply showed the entries for each field. The users felt this did not scan as well as the plain English method used to specify the conditions as seen in the conditions dialog. For this reason the lists were replaced in both the conditions and actions dialogs with a plain English description of each condition and action generated from the fields stored in the internal representation.

desirable to test more people

Only six people were used during testing due to limitations of time and practicality. A much larger test sample and longer development period would have allowed for a considerably more refined interface. However the time was not available to fully complete the implementation, and as such a larger test was neither possible nor appropriate.

Conclusion and Recommendations

Success of the project

The final implementation is still incomplete, but the implementation as it stands appears bug free, and effective. Odom’s tool was designed to be extremely flexible in its implementation, and the graphical interface places severe restriction upon that flexibility. Whilst this might appear limiting the interface is aimed at less experienced users, with little or no programming experience. As such the advanced power that is unavailable through the interface would be of little relevance or use to the target audience.

The implementation can still be utilised for:

?? Creating auto-responders for many different situations

?? Filing mail to different mailboxes depending on many different attributes

?? Handling basic spam messages.

It is accepted that in many cases some testing of the filters would be required to obtain the desired effects, this is not just a limitation of this interface, but also of Odom’s tool and indeed many other mailing packages.

The author intends to complete development of the application until such point as all the intended functionality is actually implemented. This may be because his is stubborn.

Improvements

The application could be vastly improved if it were more tightly integrated with Odom’s tool, if for example there was a predefined file-space where Odom’s tool would look for default filter files for the user, and where auto-responder messages could be stored, then this location could be placed in a configuration file for each user.

With this in place the program could offer the functionality to produce the files to be used with the ‘reply’ action within the application, and handle the file creation and modification on the user’s behalf.

This would also create scope for the current filters to be loaded into the application when it is loaded, preventing the user from having to recreate all their filters to change or add just one new filter after the current data has been processed.

It might also be beneficial to create shortcuts to commonly created filters, auto responders, spam filtering, etc. which would offer a separate wizard tailored specifically to those tasks.

More powerful condition specification would be desirable using any combination of AND/OR clauses. This is currently a difficult task to implement in a graphical environment due to the design of Odom's filter language. However improvements to this specification suggested in Odom's own report would make this task easier.

Lessons learned

Creating a fully functional user interface can be more complex than expected, very small factors affect the overall usability of an interface, and often unexpected results can occur from seemingly trivial errors.

The author has learned much about user interface design from this project, despite considerable experience in the field of web design, application interfaces require a much different approach, as the user looks for very different things in an application and an information resource.

Glossary

TO BE DONE!

Bibliography and references

ALSO TO BE DONE!!!!

Odom, NS, IE, Opera, UID…

http://www.msdn.microsoft.com/ui/

http://www.msdn.microsoft.com/library/default.asp?URL=/library/books/winguide/ch14e.htm

Acknowledgements

Lots of people, my supervisor, my friends, caffeine, comfy chairs, and pasta.

Appendix A

BNF specification of Odom's filter language

The following is the BNF specification of the scripting language developed and implemented by Odom to define filters to be used by his mail-filtering tool. This is the language that must be produced by the application this report describes.

filters ::= filter | filter filters
filter ::= **[include** filename**]**
| filter_head filter_spec
filter_head ::= **[filter** name**]**
| **[filter** name **disabled]**
filter_spec ::= **if** condition **then** actions **end**
name ::= string
condition ::= **expr**
| **NOT** condition
| **(**condition **AND** condition**)**
| **(**condition **OR** condition**)**
actions ::= action | action actions
expr ::= field match_type words | **true** | **false**
action ::= **file_message** filename
| **reply** when filename address
| **forward_message** filename address
| **pipe_to** system command
| **system** system command
| **exit**
field ::= **from** | **to** | **cc** | **reply-to** | **subject**
| **any_header** | **body** | **any** | **content** | **domain** | **sender**
match_type ::= **contains** | **equals** | **starts** | **ends**
words ::= word | word words
word ::= "string" | **{**filename**}**
when ::= **once** | **always**
address ::= **sender** | **postmaster** | sender@domain

Appendix B

Code listing