ISODE Volume 7

User's Guide

Message Handling Services

This documentation and the software are based on contributions from multiple sources. Acknowledgements for contribution are given in *ISODE Volume 1: Overview*.

This manual is associated with Version 2.0 of March 1994, and was last updated in March 1994.

Contents

List of Figures

Preface

This guide is published in support of the ISODE Consortium Release 1 (IC-R1). It is intended for users of an X.400 Message Handling Services in an OSI environment.

It is assumed that readers of this guide are already aware of the contents of *ISODE Volume 1: Overview*. Other guides to other IC-R1 products and their uses are listed under Related Publications later in this Preface.

What this Guide Covers

This guide describes tools and interfaces that provide UA (User Agent) functions for the User's Messaging Guide/ PP Message Transfer Agent.

chkmf (1): Check the *.mailfilter* syntax.

ckmail (1): Check status of your mail.

flagmail (1): Notify a server program of new mail.

mail (1): Very simple program for mail submission.

malias (1): Show what mailing lists expand to.

mlist (1): Maintain distribution lists.

ttyalert (1): Alert of new mail on the terminal.

xalert (1): Alert of new mail in an X window.

resend (1): Resubmit a message to more recipients.

strip_addr (1): Remove extraneous matter from an RFC 822 address.

mailfilter(5): Configure mail delivery.

Chapter 1 describes the interface by which messages leave the transfer system and are delivered to the user. Details of the user configurable file *.mailfilter* are explained in the context of the local RFC 822-based delivery channel. This chapter also gives information on tools for notifying the user of new message arrivals and dealing with messages for absent recipients.

Chapter 2 describes tools for the maintenance of distribution lists. The program *mlist* is used to work with file based distribution lists, while X.500 DIT entries (see *ISODE Volume 8: User's Guide: Directory Services*) can be maintained using the *DL* program.

Chapter 3 describes some simple administrative functions available to mail system users. These include a tool which can be used for checking the progress of individual messages and the validity of addresses.

Software and Documentation Versions

IC R2 is the second major release of ISODE software made by the ISODE Consortium. It is derived, with revisions and additions, from the openly available ISODE 8.0 release.

The release is accompanied by a comprehensive documentation set comprising the following volumes:

Volume	Title
01	*ISODE Volume 1: Overview*
02	*ISODE Volume 2: Administrator's Guide: Installation*
03	*ISODE Volume 3: Administrator's Guide: General Services*
04	*ISODE Volume 4: Administrator's Guide: Message Handling Services*
05	*ISODE Volume 5: Administrator's Guide: Directory Services*
07	*ISODE Volume 7: User's Guide: Message Handling Services*
08	*ISODE Volume 8: User's Guide: Directory Services*
09	*ISODE Volume 9: Programmer's Guide: General Services*
10	*ISODE Volume 10: Programmer's Guide: ASEs*
12	*ISODE Volume 12: Programmer's Guide: X.500 DUA*
13	*ISODE Volume 13: Programmer's Guide: Layer Interfaces*
14	*ISODE Volume 14: Programmer's Guide: X.400 MTA*
15	*ISODE Volume 15: Programmer's Guide: X.400 UA*
16	*ISODE Volume 16: Programmer's Guide: Transport Services*
17	*ISODE Volume 17: Programmer's Guide: ASN.1 Library*
18	*ISODE Volume 18: Programmer's Guide: ASN.1 Tools*

IC R2 is released as a number of separate modules. Each module has its own update number which is incremented independently from other modules whenever an update is made to it. The version numbers of the various components may therefore differ from one another.

As of March 1994, the latest versions of the modules and their associated documentation are as follows:

Module	Version	Volume	Last Document Update
IC Base	R2.0	01	June 1993
		02	March 1994
		03	March 1994
		09	February 1994
		10	February 1994
		13	February 1994
		16	February 1994
		17	February 1993
		18	February 1993
IC DSA	R2.0	05	March 1994
IC DUA	R2.0	08	February 1993
		12	March 1994
IC MTA	R2.0	04	March 1994
	R2.0	07	February 1994
		14	February 1993
		15	February 1993
IC SNMP	R2.0	03	March 1994
Contributed	R2.0	-	-

The ISODE Consortium

The ISODE Consortium is a not-for-profit cooperative enterprise. Its mission is to promote and evolve the ISODE implementation of OSI. This is a major package of OSI applications, including source code and documentation, with a worldwide record of operational success.

All individuals and organisations handling ISODE source code are encouraged to join the ISODE Consortium. For more information about the ISODE Consortium, contact:

Headquarters

Address:	ISODE Consortium The Dome The Square Richmond TW9 1DT UK	Phone: Fax:	+44 81 332 9091 +44 81 332 9019

Email

Internet (RFC822)ic-info@isode.com

MHS (X.400): S=ic-info; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI;

If this is not possible, you may also reach the ISODE Consortium at:

US Office

Address:	ISODE Consortium PO Box 200195 Austin TX 78720 USA	Phone: Fax:	+1 (512) 338 3340 +1 (512) 338 3600

Availability

IC R2 is a source release, and is available in the following ways:

1. To organisational members of the ISODE Consortium as a benefit of membership.
2. To educational and non-commercial research organisations, under zero-cost licence from the ISODE Consortium or under licence from a Research Consortium member of the ISODE Consortium.
3. To any organisation as a 30 day evaluation copy from the ISODE Consortium.

Acquisition and use of IC R2 and related materials for any purpose requires a written licence agreement from ISODE Consortium Limited, or a written licence from an organisation licenced by ISODE Consortium Limited to grant such a licence.

Binary products based on IC R2 are available on a wide range of platforms from ISODE Consortium members. Information on these products and where to obtain them is available from the ISODE Consortium.

Bug Reporting

Bug reports on software releases made by the ISODE Consortium are welcomed. These may be sent to the Consortium by any means, but electronic mail to one of the bug reporting addresses is preferred. Please send proposed fixes with the reports if possible. For non-members of the ISODE Consortium, any reports will be acknowledged, but no further action is guaranteed. Any changes resulting from bug reports may be included in future releases.

The bug reporting addresses in Internet (RFC 822) format are:

- `bug-isode@isode.com` for general ISODE reports.
- `bug-quipu@isode.com` for reports on QUIPU (Directory).
- `bug-pp@isode.com` for reports on PP (MHS).
- `bug-snmp@isode.com` for reports on SNMP.
- `bug-doc@isode.com` for reports on the documentation.

The MHS (X.400) equivalents of these addresses are:

- `S=bug-isode; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-quipu; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-pp; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-snmp; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`
- `S=bug-doc; O=ISODE Consortium; P=ISODE; A=MAILNET; C=FI`

Support

Basic support of ISODE Consortium releases is available to all consortium members as a benefit of membership. This includes upgrades and access to a database of bug reports and fixes. The ISODE Consortium also offers some additional support to companies building products specifically based on ISODE Consortium releases.

A range of ISODE-related support services are offered by ISODE Consortium members. A list of such services will also be available from the ISODE Consortium.

Chapter 1

Local delivery in PP

The local channel is the mechanism by which PP delivers mail to users. There may be several of these channels tailored to specific mailbox formats. A user may have more than one local channel associated with him/her where this is useful, such as having two formats of mail (e.g., X.400 and RFC 822) delivered in different ways.

1.1 Local RFC 822-based Channel

The RFC 822 local channel has considerable control over the delivery of the message. In particular, the channel has a particular search path which is followed when the message is being delivered. If the message is considered delivered, or rejected by any one of these methods, then the further methods of delivery are attempted. The later methods of delivery are only attempted if the earlier ones are inconclusive. (This does not apply to the *.mailfilter* processing, which normally processes the entire file.)

1. If the user has a *.mailfilter* file in his/her home directory and the file is owned by that user and not writable by anyone other than that user, the commands within that file are obeyed. The syntax is defined in Section 1.1.1. If the user does not have a *.mailfilter*, or its actions do not cause delivery, rejection, or temporary failure of the message, the following method is attempted.
2. If there is a system defined *.mailfilter*, then the commands in that are obeyed. Again if this does not exist, or the actions are inconclusive, the following action is taken.
3. If if neither of these two methods result in the message being delivered, it is surrounded by delimiters and written to a file in a well-defined (system configured) place.

1.1.1 Format of the .mailfilter File

The *.mailfilter* file allows quite complex processing of incoming messages to be performed. The format takes the form of a small interpreted language in which expressions can be tested and evaluated prior to performing actions. In particular, conditional actions are the most useful.

The language is defined more formally in the grammar given in Figure 1.2. However, a more gentle introduction is given here with some examples of usage.

Commands

There are two basic commands that can be performed on a message. These commands can be performed any number of times, so the message can be delivered in several places and/or to several processes.

The first command is `file`. This command takes an argument which is a string and the result is to attempt to copy the message surrounded by delimiters into the given file. This might be as in one of the following examples:

```
file "mailbox";
file "/archive/mailbox";
```

Note that all statements in the *.mailfilter* file must be terminated with a semi-colon ("`;`").

A variant of the `file` command is the `unixfile` command. This writes into the given file, but in sendmail or traditional UNIX-style format; i.e., with a header line stating `From`.

```
unixfile ".mail";
```

The second basic command is `pipe`. This command starts up a process and writes the message to its standard input. This is shown in the next example.

```
pipe "rcvstore +inbox";
pipe "lpr -p -Ppostscript";
```

The system can tell if these commands have succeeded by the following methods.

If the file writing works, then a flag `delivered` is set to indicate success.

Similarly, if the exit status from a `pipe` is zero, the delivery is considered a success and the flag updated.

If a program returns a "temporary failure" by exiting with a value between 1 and 127, the delivery attempt is aborted and retried at a later date.

An exit status greater than 127 is taken as permanent failure.

Sometimes, though, you may wish to run a command and ignore the results; that is, to have the system not update the status of the delivery. This can be achieved by setting a variable as described later, but for simplicity, if you precede the command with the keyword `ignore`, then no notice will be taken of the return status.

```
ignore pipe "echo hello";
ignore file "/dev/null";
ignore pipe "rcvalert 'you have new mail'";
```

One last feature of these commands, header variables (described below) may be substituted in the strings associated with these commands. These are not too useful for file commands, but can be useful as arguments for processes. Care must be taken that appropriate quoting of these is done in case they contain special characters. As any string containing a "$" is substituted, a $ must be inserted by doubling it, e.g., "$$". If you require a quote """ character in the string, it must be preceeded by a backslash character: "\".

Normally, the commands are interpreted by /bin/sh -c <command> but if the *822-local* channel is running in restricted mode, the command is directly executed.

Conditional Expressions

The real power of the *.mailfilter* comes with conditional expresssions. These allow the user to select particular rules depending, for instance, on certain attributes of the message.

The most basic form of condition is an if statement with a simple comparison. This allows you to compare two strings for equality. This takes the form of one of the following:

```
if ( <string1> == <string2> )
    <expression> ;

if ( <string1> == <string2> ) {
    <expression1> ;
    <expression2> ;
}

if ( <string1> == <string2> )
    <expression> ;
else if ( <string1> == <string3> )
    <expression> ;
else
    <expression> ;
```

These examples show a number of things. First, where you can have a single expression, you may have several expressions by enclosing them in braces[1]; this is an example of a compound expression.

Second, by use of the else statement, default actions can be taken. Finally, the else if allows a multi way branching construct.

Conditional expressions may also contain Boolean operators. The list is described below.

Operator	Meaning
==	Is equal to
!=	Is not equal to
&&	Logical and
||	Logical or
!	Logical not

The conditional construct begs the question "what conditions can be tested?" These are discussed in the next section.

[1]Any resemblance to the *C* programming language is entirely intentional!

Variables

There are two basic types of variables that can be accessed:

1. System variables. These are variables maintained by the *822-local* channel and may be updated after actions are taken.
2. Header fields. When delivering a message, all the header fields that are current are available for examination.

The system variables are the simplest and there are very few of these. The most used system variable is `delivered`. This variable normally contains the status of the delivery attempt. This is what the system thinks the state of your message is. This may be one of the following values.

Value	Meaning
`TRUE`	The message has been delivered
`OK`	The same as above
`FALSE`	The message has not been delivered
`PERMFAIL`	The message has permanently failed to be delivered. A delivery report should be sent back if required.
`TEMPFAIL`	The message has failed to be delivered for some temporary reason. Another attempt may succeed.

This may be used in conditional expressions to test for delivery, or in expressions to reset the delivery state. For example:

```
if ( delivered == FALSE) file "foo";

if ( delivered != TRUE ) file "bar";

# If not delivered - reject the message
if ( delivered == FALSE) delivered = PERMFAIL;
```

Another system type variable is `PATH`. If this variable is assigned a value, then it is placed in the environment as the default path. This is used to search for commands when executing `pipe` commands. It is initially set to nothing and the user's shell default search path is used.

Note: This variable is not used if the `822-local` channel is configured in restricted mode.

The second class of variables, which are more useful for automatic filing and delivery, are the variables derived from the messages header. These variables evaluate to the contents of the particular field, or to the null string if there is no such variable. These variables can be compared against quoted strings, but such usage is not generally useful. These variables are distinguished by always starting with a dollar symbol ("`$`") followed by the field name in parentheses. (e.g., `$(to)` or `$(from)`). Such comparisons require exact matches, including whitespace, with the exception that letter case is not significant.

Much more useful is comparison against a regular expression. The regular expressions used are those in *regexp* (3). These match, by default, against substrings of the fields. More complex patterns can also be built up. Regular expressions, which must always appear on the right hand side of the conditional expression, are delimited by slash marks ("`/`"). An example of such usage shows the real power of the *.mailfilter* file:

```
PATH="/usr/local/lib/mh:/usr/local/bin:/usr/bin:/bin:/usr/ucb";
folder = FALSE;
if ($(to) == /jpo/ || $(to) == /onions/) {
     pipe "rcvstore +to"; folder = "to";
} else if ($(sender) == /bind-request/) {
     pipe "rcvstore +bind"; folder = "bind";
} else if ($(sender) == /mailgroup/) {
     pipe "rcvstore +janet/mailgroup"; folder = "janet/mailgroup";
} else if ($(sender) == /namedropper/) {
     pipe "rcvstore +mail/namedroppers"; folder = "namedroppers";
} else if ($(sender) == /pp.*-request/) {
     pipe "rcvstore +pp"; folder = "pp";
}
if (!delivered) {
     pipe "rcvstore +inbox";
     if (!delivered) {
           ignore pipe "flagmail -f 'DELIVERY FAILED'";
           exit (TEMPFAIL);
     }
     folder = "inbox";
}
if (folder)
     ignore pipe "flgamail -s $(size) -e '$(folder): '";
```

Figure 1.1: Example .mailfilter File

```
if ( $(from) == /namedroppers/ ||
     $(from) == /bind/) pipe "rcvstore +mailadmin";
else pipe "rcvstore +inbox";
```

Finally, there are some variables that may appear to be header fields, but are actually derived from other sources. These are:

`size`	The size of the message in bytes as a string.
`return-path`	The original sender of the message.
`mailbox`	The mailbox file associated with this user.
`recipient`	The address of this recipient.
`userid`	Either the username or userid of this recipient.
`groupid`	The group id of this recipient.
`channelname`	The name of this delivery channel.
`hostname`	The name of the originator's host.
`opts`	The value of the `opts` field from the delivery channel table.

Other Features

There are a few other features that can be used in the *.mailfilter* file. These are summarised below:

`exit`: This statement exits the processing of the *.mailfilter* file. With no argument it exits with the current value of the `delivered` variable. If an argument is given, then that is taken as the success or failure of the delivery attempt. The argument should be one of `OK`, `TEMPFAIL`, `PERMFAIL` or a variable that evaluates to one of these.

`print`: Normally, this statement does nothing and is ignored. However when running the checking program, it will echo its arguments on the standard output. This may be useful in debugging.

`#`: This symbol starts a comment, which finishes at the end of the line.

1.1.2 Examples

The following are some simple examples of *.mailfilter* files.

This first example shows just a simple delivery making use of the MH *rcvstore* (1) utility:

```
pipe "rcvstore +inbox";
```

The next example shows slightly more complex behaviour, whereby some mailfiltering occurs, and an alert is posted:

```
if ($(to) == /jpo/)
pipe "rcvstore +to";
else
pipe "rcvstore +inbox";
ignore pipe "flagmail -s $(size)";
```

A more complex example of a *.mailfilter* file is shown in Figure 1.1.

The grammar is specified in Figure 1.2 on page 13.

1.2 Mailfilter Checkup Tool

As writing *.mailfilter* files can be error prone, a tool to check that the contents conform to the language is provided. This is invoked with a given filename and a message on the standard input. The file is parsed and then interpreted and the actions appropriate for the given message are indicated.

```
ckmf ~/.mailfilter < msg
```

It is *strongly* recommended that you check your *.mailfilter* file each time it is modified. This will certainly pick out syntax errors and the like, however, it will not discover errors such as the following which may be common causes of problems:

- Unable to find the program to be run (not in search path).
- The process that is executed aborts. This may be because some things in the environment are missing. If this is the case it is best to wrap the process in a shell script. Alternatively, the process may expect the standard input/ouput be connected to a terminal. This is difficult to get around.

An example run of *ckmf* might look like this:

```
john% ckmf ~jpo/.mailfilter < /dev/null
pipe into process 'rcvstore +inbox'
pipe into process 'flagmail -s 0 -e 'inbox: ''
Result: delivered OK
john%
```

1.3 New Message Notification

PP provides a couple of simple tools for alerting the user of new messages arriving. Both methods share a lot of common features (and code, for that matter).

The basic idea of the notification is to run a process from the delivery process which reads the message, extracts some information, formats it, and then gives it to the user. This may be either by displaying this formatted information on the user's terminal, or on a client of an X Window System[2] display.

There are four processes involved. They are:

`xalert`: This process runs as a client of an X windowing system server. It displays messages sent to it by *flagmail*. If it detects that the X windowing system is not in use, it captures messages and displays them on the terminal.

`flagmail`: This process gathers the data from the message, formats it and attempts to pass it to *xalert* (or a similar process obeying the same protocol). If it fails it emulates the *ttyalert* program.

`ttyalert`: This process gathers data from a message and simply displays it on the user's terminal if logged in. It is only intended for large sites where all people use the same host.

`ttybiff`: This process runs in the background normally. It displays messages sent to it by *flagmail*.

This program's use is deprecated as the functionality is subsumed by *xalert*. It will probably not appear in future releases.

Both *flagmail* and *ttyalert* use the same method of preparing and formatting the data to be displayed. This is done by using a format string to indicate how the information should be laid out. The format string can contain verbatim text, header fields, and other attributes of the message.

Fields within the format string are indicated by percent (`%`) symbols. The general format bears some relationship to the *printf* (3) format:

```
%[digits]string
```

The digits are optional and indicate how large the field should be padded out to or truncated to. The string is any message field, or one of a set of special fields. The case of the string is not sensitive.

[2] The X Window System is a trademark of the Massachusetts Institute of Technology

The special fields are:

Field	Meaning
`body`	The first few characters of the body of the message
`size`	The size in bytes of the message
`extra`	A special field set on the command line
`from`	the from field is treated in a special way

The `body` field is the first few characters of the body with all newlines removed and multiple spaces compressed to a single whitespace.

The `size` is set on the command line by the use of an optional flag. This is usually done in the *.mailfilter* file by use of the `$(size)` variable.

The `extra` field is used to allow simple changes to the default format string. This just allows a simple flag to adjust the output from the default format string rather than rewriting the format string. If not set, it defaults to the value `NEW: `.

The `From` field is treated slightly specially, in that if there is a name component in front of the opening angle bracket, then the rest of the field is ignored.

The default format string is the following:

```
%extra(%size) %20from %subject << %50body
```

This gives output something like the following

```
NEW: (3402) Steve Kille  Re: state of unix-niftp? << I
```

The whole output line is truncated to the number of characters set on the command line (defaulting to 80). This allows the later parts of the string to be large, in the assurance that they will get chopped off if they exceed the given size.

An example usage in a *.mailfilter* file might look like the following:

```
if (delivered)
    ignore pipe "flagmail -s $(size)"
```

See the example in Figure 1.1 on page 5 for a more complex example of usage.

Flags

Both *flagmail* and *ttyalert* respond to a number of command line options to change the behaviour. These are:

`-d`: Runs the program in a debugging mode. This stops the program from running in the background. Only useful if you are trying to debug the program.

`-e string`: Sets the `extra` variable. This is used to override the default extra information which is normally `NEW:`.

`-f format`: Sets the entire format string to the given string. This is how you override the internal format string with something entirely different.

`-s size`: Sets the message size parameter.

`-w width`: Sets the default output line width. The line produced will be truncated to this width (default 80 characters).

`-x`: Forces the use of protocol. In this mode it will not try and do its own notification if it fails to notify a server.

In addition the *flagmail* program has two other options to tailor the interaction with *xalert* or other servers.

`-h hostfile`: Use the given file to read the host and port to communicate with. This overrides the default file (*.alert*) in the home directory.

`-p number`: Use the following number to override the UDP port found in the *.alert* file.

`-a address`: The full address of the alert process with which to communicate. It has the format `hostname[:port]`.

`-u user`: Change the idea of the user to whom the alert should be sent. If used with the `-a` flag then neither the `.alert` file nor password database (`/etc/passwd`) need be consulted.

Some command line switches are specific to *xalert*, namely:

`-filename hostfile`: use the given file to write the host and port number in instead of the default file (*.alert*).

`-user username`: *xalert* selects requests destined for the given user. Normally it will not receive messages destined for another user, but this will change *xalert*'s idea of the current user anyway.

`-port number`: Set the UDP port number to the given number. The default is to choose a random unused UDP port.

`-s`: Only print to the standard output; do not display on all this user's login sessions.

`-t`: Do not attempt to use the X system.

`-x`: Allow use of X (default).

`-w width`: Use this width when displaying messages on the terminal.

`-bell/-nobell`: Notify (or not) new message with a bell noise (the default is `-bell`).

`-autoraise/-noautoraise`: Raise (or not) the window to the top of the stack when new messages arrive. The default is not to raise the window.

`-photos/-nophotos`: Display a photograph of the sender of the mail on reception. This depends on whether the code is compiled to look up the entry in the directory.

In addition it takes all the standard X command line arguments.

ttybiff supports the following flags similar to *xalert*:

`-f hostfile`: use the given file to write the host and port number in instead of the default.

`-u username`: as for *xalert*'s `-username` switch.

`-p port`: as for *xalert*'s `-port` switch.

`-s`: only display on the standard output, not on other terminals where the user is logged on.

1.3.1 Use of flagmail with xalert and ttybiff

Xalert works with *flagmail* or *ttybiff* to provide the notification. *Flagmail* is responsible for building up the format string and sending it to either *xalert* or *ttybiff*. This is achieved using a UDP connection. *flagmail* contacts the server program by looking for a file with a hostname and UDP port in (by default the *.alert*). If this exists then it attempts to contact a server on that host to display the message. If either of these steps fails, then it reverts to the behaviour of *ttyalert* except that there is a delay while it waits for UDP timeouts to occur.

The *.alert* has the format

```
<hostname> <udp-port-no>
```

flagmail will use this information to contact the remote process, which can be any process which understands the *flagmail* protocol.

The *xalert* program is just a display process that displays the messages sent by *flagmail*. It displays these message in a text widget which is scrollable.

The right button pops up a menu with a number of options:

`Clear`: Clears the message window.

`Reset`: Rewrites the *.alert* file; it is used if two xalerts are started or the *.alert* file is removed.

`Bell`: Disables/enables use of the bell. The default can be overriden in the users X resource file by setting a boolean value for `XAlert.bell`.

`Autoraise`: Disables/enables the autoraise mode. The default for this can be set via the boolean X resource `XAlert.autoRaise`.

`Photos`: Disables/enables display of photographs. The default for this can be set via the boolean X resource `XAlert.photos`.

`Quit`: Does the obvious thing.

The *ttybiff* attempts to locate the user in the same manner as *ttyalert* or, if that fails, to display the message on the standard output.

1.3.2 Examples

Here are a few examples of use.

First, the simple notification that should be put in the *.mailfilter* file:

```
ignore pipe "flagmail -s $(size)";
```

This does the most basic notification. If more control is required over the displayed message more complex strings can be built up:

```
if ($(subject) == /request/) {
     file "request-mbox";
     ignore pipe "flagmail -s $(size) -e REQUEST";
}
```

Finally, if you are adventurous, you might like to play with the format of the message given. For instance:

```
ignore pipe
 "flagmail -s $(size) -f '%5size %15from (%20subject) **%body'";
```

This would give an output in the following format for a typical message

```
1023  Julian Onions   (Re: Delivery Report ) ** Also
```

1.4 Automatic Vacation Notices

PP provides a simple program to notify originators that the recipients will not read their messages for some time. This is typically installed when the recipient is expecting to be unable to read mail for some time, such as going on vacation.

The program, *rcvtrip*, is installed in the *.mailfilter* file in a fashion similar to the following.

```
ignore pipe "rcvtrip"
```

The program reads the message and checks to see if the recipient is mentioned directly in either the `to` or `cc` line. If so, and the originator has not been replied to in a similar style before, then a reply is composed. This consists of some standard text, followed by the contents of the file *tripnote* in the user's home directory.

There are several flags to this program.

`-F`**:** Always reply to the originator, even if the recipient was not named directly on the original mail.

`-e value`**:** Set the exit code depending on this value. A value of 0 means the program always exits with failure (the default). If the value is 1, the exit code is always good. A value of 2 will give an exit status according to the following table:

Condition	exit code
A message was sent	1
A message was not sent as this sender has already received a message	2
A message was not sent as this message was not sent directly to this user	3
A parse error was found	4
Anything else	99

`-f`**:** Set the text of the from field (defaults to `MESSAGE AGENT`).

`-l file`**:** Set the database to this filename (default *triplog*).

`-n`**:** Use this file as the text of the message to send back after the standard preamble (default *tripnote*).

`-s`: Use this file to determine the user's signature (default *.signature*).

`-t text`: Modify the preamble to use this line (several of these may be given).

`-u user`: Pretend to be this user when answering.

1.5 Forwarding Mail

There is a simple program which will forward mail to other recipients. This is called *resend*. This program takes a message as standard input with extra recipients given on the command line. The program will then submit the message as a `resent` message. The new recipients will appear in the `resent-to` and `resent-cc` lines as appropriate.

There are two flags that control the operation of this program:

`-t`: The next recipients should be specified as `Resent-To:` recipients.

`-c`: The next recipients should be specified as `Resent-Cc:` recipients.

`-u username`: Pretend to be this user (subject to authorisation checks).

1.6 Delivery Tools

There are two small tools that are provided to help with delivery processing.

1.6.1 strip_addr

The *strip_addr* utility takes a list of addresses either on the command line or on the standard input and issues those same addresses in a basic form. In particular, all address comments are removed, as are name and list constructs. This utility may be useful in processing mail where only the basic address is wanted.

strip_addr takes two optional flags. These are:

`-p`: Interpret the `%` character as a routing construct. If this flag is not present, the `%` character will be taken as just another character.

`-r`: Strip out routes as well.

An example of use might be:

```
strip_addr 'Julian Onions <jpo(thats me)@there.com>'
jpo@there.com
```

1.6.2 flock

The *flock* utility takes a single file followed by a command. The file is first locked in the current PP locking style, and then the command is executed. When the command exits, the file is unlocked and *flock* exits with the returned status of the command.

```
<statementlist> ::= <statement>
                  | <statement> <statementlist>

<statement> ::= <expression> ";"
                  | ";"
                  | "print" <prexp> ";"
                  | "if" <condition> <statement>
                  | "if" <condition> <statement> "else" <statement>
                  | "{" <statementlist> "}"

<condition> ::= "(" <expression> ")"

<expression> ::=  "(" <expression> ")"
                  | <expression> "==" <regex>
                  | <expression> "!=" <regex>
                  | <variable> "=" <expression>
                  | <expression> "==" <expression>
                  | <expression> "!=" <expression>
                  | <expression> "||" <expression>
                  | <expression> "&&" <expression>
                  | "!" <expression>
                  | [ "ignore" ] "file" <string>
                  | [ "ignore" ] "pipe" <string>
                  | "exit" [ <expression> ]
                  | <field>
                  | <variable>
                  | <string>

<prexp> ::= <expression>
                  | <prexp> "," <expression>

<string> ::= quoted string of characters (e.g., "abced")

<variable> ::= list of characters of the form [a−zA−Z][a−zA−Z0−9]*

<field> ::= "$(" list of characters and − ")"

<regex> ::= "/" regular expression (see regexp(3)) "/"
```

Figure 1.2: Grammar for the .mailfilter File

Chapter 2

Distribution List Maintenance

2.1 File based Distribution Lists

PP provides an interactive tool for maintaining distribution lists, *mlist*. This tool allows you, the user, to view, modify and generally manage the members of a given list. What you are allowed to do to a list depends on the access rights set on that list.

The access rights for a list are set when the one of the managers of that list first uses *mlist* to manage the list. There are four modes of access rights:

free: Anyone can add or remove members of the list.

public: Anyone can add themselves to or remove themselves from the list.

private: Only managers of the list can add or remove members;

secret: Only managers of the list can view or alter the members of a list.

If you are a manager of a list, the access rights are bypassed and you are given *free* access to the list.

2.1.1 What you can and cannot do with *mlist*

For all modes of access except *free*, the maintenance of the list is done via a series of questions. For example:

```
You (pac) are not in this list
Do you wish to be added ? [y/n]
```

With *free* access, the maintenance is done via a menu of options.

add: A list of possible mail addresses is expected with this option (either following the add command or on the next line). *mlist* checks that each address is a valid mail address and that the normalised address does not already exist in the list. Addresses that pass both these tests are added to the list.

remove: A regular expression is expected with this option. Any address that matches this expression is a candidate for removal from the list. *mlist* asks the user whether each candidate should be removed.

find: As with the *remove* option, a regular expression is expected. Any address that matches the expressions is printed out.

verify: This option goes through the list checking that each member is a valid mail address.

As well as these maintenance options, there are a few inspection options

print: This option prints out all the information concerning the list. This information consists of all the members of the list, the list moderator's mail address, and the user ids of the moderators of the list.

list: This option displays all the lists known to the local PP system. Each list also has a short description of its purpose.

help: This option displays all the options that are available to you.

If the list you require does not exist, there is one further option, `create`. This option prompts you for all the details of the list you require and posts them to the postmaster. If the postmaster decides to grant you this list, he or she will then do the appropriate administration to create the list. Once the postmaster has done this, you can use *mlist* to manipulate the list.

If you have done this and the list you requested now appears in the output of the `list` option but *mlist* still thinks it does not know about that list, then it is possible that the postmaster overlooked creating an empty file to contain the list when he or she was adding the list to PP's tables.

2.1.2 How *mlist* can run

mlist takes each list specified on the command line in turn, and enables you to modify it. If no lists are specified on the command line, *mlist* enters a loop. Within this loop, you choose the lists you wish to modify. When you have finished modifying the required lists, you can choose to `quit` the loop and exit *mlist*.

mlist can take several command line arguments:

-user 'username' This specifies the username one wants to be known as when altering lists. Only privileged users can use this flag.

-uid 'userid' This specifies the user Id one wants to be known as when altering lists. Only privileged users can use this flag.

-order 'domain order' This specifies the preferred domain ordering to use when parsing addresses. This can be specified as:

- **usa** usa domain ordering ONLY.
- **uk** uk domain ordering ONLY.
- **usapref** usa domain ordering preferred.
- **ukpref** uk domain ordering preferred.

The default is usa domain ordering only.

2.1.3 *malias*.

The *malias* tool is a passive version of *mlist*. Using the *malias* tool, you can only view the members of specified lists. You are unable to modify, in any way, the contents of a list.

2.2 X.500 Directory based distribution lists

Directory lists use the OSI directory to store the list of recipients. This has the advantage that the list can be easily viewed using any Directory User Agent (DUA), making the management task easier. Some knowledge of the QUIPU Directory service is assumed.

2.2.1 The Basics

The manager of the system must set up a directory list in the user table for PP to be able to expand them. Once this is done, the maintenance of the list can be delegated to anyone with the appropriate permissions. To set up a local directory based list called `raboof`, an entry in the *users* table is needed. This should look something like

```
raboof:dirlist lancaster.xtel.co.uk
```

Each mail list needs a manager. Typically this is the mail list name with `-request` appended, `raboof-request` in this case. So the PP *aliases* table may contain something like:

```
raboof-request:alias c.robbins
```

When mail for `raboof` comes in, the directory list channel is run The first action is to convert the `raboof` name into a Directory Name. First the ISODE *isoaliases* in the *ETCDIR* is consulted. If there is an entry of the form

```
raboof  "c=GB@o=Site@cn=The Raboof List"
```

Then the given directory name is used. If no such entry is found, then a one level directory *search* operation is used. The base of the search is defined by the `local_DIT` variable in the *dsaptailor* file in the *ETCDIR* directory. The search looks for entries belonging to the objectclass `ppDistributionList` with a `commonName` attribute of `raboof`. [1]

Finally, the entry is read and the mail is delivered to the indicated recipients.

2.2.2 The DIT Entries for the list

The main entry, as described above must belong to the object class `ppDistributionList`. This has the following mandatory attributes:

[1] `raboof` does not have to be the RDN of the entry, just an attribute. However you should ensure that the above search will only match one entry. If in doubt use an alias.

`mhsDLMembers`: The ORName of members of the list. Section 2.2.3 discusses the format of ORNames in detail.

`owner`: The DN of the owner of the list. Typically, this will be a role entry for the `-request` of the list. The object class `ppRole` is useful for this, as it allows you to specify an `mhsORAddress` for the role. A typical entry might be:

```
cn= The Raboof List Manager
cn= raboof-request
roleOccupant= c=GB@o=X-Tel Services Ltd@cn=Colin Robbins
mhsORAddresses= /S=Raboof-request/O=XTEL/PRMD=X-Tel Services/ADMD= /C=GB/
```

`cn`: The name of the list.

`mhsDLSubmitPermissions`: This defines who can send to the list. The default is a special token "`ALL`" which permits anybody to send mail to the list. However, it is possible to restrict the use of the list. The syntax is:

```
<permission> ::= <group> | <individual> | \
                 <member> | <pattern>
<group> ::= 'GROUP #' <DN>
```

A member of the X.500 group `DN` can send to the list.

```
<individual> ::= 'INDIVIDUAL #' <ORName>
```

The individual defined by `ORName` may send to the list.

```
<member> ::= 'MEMBER #' <ORName>
```

A Member of the named list may send. This only works if the named list is also directory based.

```
<pattern> ::= 'PATTERN #' <ORName>
```

Anybody whose ORName Matches the <ORName> pattern can send. E.g., if <ORName> was "c=GB", then only people in "GB" could send to the list.

`mhsORAddresses`: The mail address of the list, in this case

```
/S=raboof/O=XTEL/PRMD=X-Tel Ltd/ADMD= /C=GB/
```

The optional attributes are listed below.

`dl-policy`: The list delivery policy. It is generally best to leave this to the default value of: further list expansion allowed; use the original parameter for the conversion prohibited header; use low priority.

To change the default the syntax is

```
<policy> ::= <expansion> $ <conversion> $ <priority>
<expansion> ::= 'TRUE' | 'FALSE'
```

If `TRUE` sub lists are also expanded.

```
<conversion> ::= 'ORIGINAL' | 'TRUE' | 'FALSE'
```

Is message content conversion allowed. `ORIGINAL` implies use the parameter from the original message.

```
<priority> ::= 'ORIGINAL' | 'HIGH' | 'LOW' | 'NORMAL'
```

The priority to use when delivering the message. `ORIGINAL` implies use the parameter from the original message.

`description`: A textual description of the list.

The following attributes may be present but are not used by the directory list channel.

`o`: The associated organisation (if any)

`ou`: The associated organisationalUnit (if any).

`seeAlso`: A reference to another directory entry.

`mhsDeliverableContentTypes`: Acceptable content types.

`mhsdeliverableEits`: Acceptable encoded information types.

`mhsPreferredDeliveryMethods`: A list of preferred delivery methods.

2.2.3 ORNames

The most important part of directory lists is the originator-recipient name (ORName) of members of the list. An ORName consists of an optional directory name (DN), a dollar ("`$`") separator and an originator-recipient address (ORAddress).

ORAddresses can be specified as text encoded X.400 addresses

```
/S=Robbins/O=XTEL/PRMD=X-Tel Services/ADMD= /C=GB/
```

or as rfc822 mail addresses

```
C.Robbins@xtel.co.uk
```

In the later case, this will be converted into a X.400 style address. This conversion is useful as a short cut in giving all users with a `rfc822mailbox` attribute in the directory a `mhsORAddress` attribute.

To specify an ORName as an ORAddress only (with no DN component) you need to precede the ORAddress with a "`$`" symbol [2]. So, the above ORAddresses would become

```
$ /S=Robbins/O=XTEL/PRMD=X-Tel Ltd/ADMD= /C=GB/
```

and

```
$ C.Robbins@xtel.co.uk
```

as ORNames. It the "`$`" is omitted, the interfaces will try to reverse map the mail address onto a DN — If the user does have a directory entry, it often succeeds!

It is much preferable to use a DN instead of a mail address. The DN component can be specified in one of two ways, either as a fully qualified, QUIPU style DN:

```
@c=GB@o=X-Tel Services Ltd@cn=Colin Robbins
```

[2] If the programs were smart enough, this could probably be omitted in most cases, but it has the advantage of making the user think, and hopefully use a DN or UFN instead

or as a UFN

```
Robbins, xtel, GB
```

Neither of these examples give an ORAddress. The interfaces will determine this for themselves using the directory. If you do want to specify an address, then you can use the "`$`" separator:

```
Robbins, xtel, GB $ C.Robbins@xtel.co.uk
```

2.2.4 The DL tool

The program DL can be used to create and maintain the directory entries for list management, and is a very useful companion to the PP enhanced version of DISH.

To invoke DL use the *dl* command. It will read your *.quipurc* to obtain your directory username and password. The username can be changed with a "`-u`" flag, you will be prompted for a password. As with DISH you can request connection to a different DSA with the "`-c`" flag.

DL attempts to interpret what you can do to the list in terms of your access rights, based on the DN you bound with. If you are the DSA manager, you can use the "`-m`" flag to tell DL you have full access rights.

On entry to DL, typing a "`?`" will offer some limited help.

When you enter DL you will be placed at the default position in the DIT, as defined by the variable `local_DIT` in the *dsaptailor* file. You can alter this using the "`m`" command, and supplying a UFN:

```
> m
Currently at 'X-Tel Services Ltd'
Enter new location: cs, ucl, gb
Searching...
Found exact match(es) for 'GB'
Found good match(es) for 'ucl'
Found good match(es) for 'cs'
Moved!
>
```

You can find out which lists exist at a given level in the DIT with the "`l`" option:

```
> l
Found the following lists:
  GB Directory Pilot
  JNT PP
  JNT Quipu
  X.500 Pilot Support
  xtel
>
```

The "`w`" option will tell you which of those lists you are on.

2.2.5 Creating a List

A list can be created in one of two ways. Either an existing PP file based list can be upgraded, or a list created from scratch.

If you are bound to the directory as a "normal" user you will not have access rights to modify the directory. So for both upgrading and list creation, the DL program will ask the user a few simple questions and then send a mail message to the postmaster.

If you are bound as the DSA manager (using the "`-m`" start up flags to DL), the DL program will attempt to make the necessary changes to the directory for you.

Creating

To create a list use the "`c`" command. You will be prompted for the name of the list, the owner of the list (UFN), and textual description.

DL will create `listname-request` and `listname` entries in the directory.

Finally DL advises you to alter the PP tables to add the new list.

Upgrading

Upgrading is achieved using the "`u`" option in the DL program. You will be asked for the name of the PP file based list.

DL will create `listname-request` and `listname` entries in the directory. The members of the list will be read from the file based list, and added into the directory list. As many as possible will be reversed mapped into directory names.

Finally DL advises you to alter the PP tables to change from a `list` to a `dirlist`.

2.2.6 Managing a List

The DL program can be used to manage a list after it has been created. To manage the `raboof` list you can either invoke DL with the list name as an argument

```
% dl raboof
raboof>
```

or run DL then give the list name

```
% dl
> raboof
Reading 'raboof'...
raboof>
```

In either case the new prompt will contain the list name. A "`?`" can be typed to obtain limited help.

To add a member, simply enter the ORName, using the syntax described in Section 2.2.3 at the "listname" prompt.

The "`p`" option is used to print the members of the list. If a DN is present, *only* the DN is printed. Each entry is preceded by a number. This number can be used to reference the entry when using the "`r`" — remove entry option, for example:

```
raboof> p

Owner: Colin Robbins

Submit Permissions,   Entities matching the OR pattern:
   X.400 Address: /C=GB/

Entry has no dl-policy

Members:
 1: Steve Hardcastle-Kille, Computer Science,
   University College London
 2: Colin Robbins, X-Tel Services Ltd
 3: Julian Onions, X-Tel Services Ltd

jnt-quipu> r
give username to be removed: 2
Remove Colin Robbins [y/n/d] ? n
no
```

The "`d`" option in the "`[y/n/d]`" prompt is used to display the directory entry ("`y`" for yes and "`n`" for no).

Alternatively an ORName could be given, but as UFN is invoked, it may take too long to find to correct user. Using numbers is quicker.

Attributes such as the list policy, and submission policy can be changed using the "`m`" option.

2.2.7 Maintaining a List

From time to time mail addresses will change. One of the advantages of the directory list is that such changes can to some extent be tracked [3] as long as the directory is kept upto date.

When accessing a particular list, the "`c`" and "`v`" commands to the DL program can be used to check or validate the list. The entry for each member of the list is checked for consistency. Each ORAddress is check to see if the local PP system thinks they are valid. If the ORName has a DN component, the given ORAddress is checked against the address given by the directory. Using the "`c`" option errors are reported, using the "`v`" an attempt is made to update bad ORAddresses.

You are advised to run "check" or "validate" on your lists regularly. Checking the lists can be done from *cron* using the "`-v`" option, for example:

```
dl -v raboof
```

This will warn (verbosely) of errors. The "`-q`" option is useful to quieten DL down.

You can check all lists this way, by not specifying a list name, for example:

```
dl -v -q -m
```

will (quietly) check all lists. The "`-m`" flag is used here to tell DL it is in management mode. Normally, the invoker of DL will be told of the errors in the list. When run at a site with a lot of lists,

[3] If the DN changes you are still in trouble!

each with different managers, it is often more useful to mail the list managers of the errors, rather than mail the invoker of DL.

2.2.8 Printing a List

When DL prints the members of a large list, a pager such as *more* will be used to prevent the list scrolling past. Sometimes this can get in the way, for example if you want to pipe the output onto a printer. The "`-p`" flag to DL can be used to prevent this e.g.,

```
dl -p raboof
```

will cause DL to print the raboof list the the standard output, then exit.

Chapter 3

Miscellaneous Utilities

There are a number of small utility programs provided with PP that may be of interest to end users. These are described here.

3.1 A /bin/mail Replacement

PP comes with a very simple user interface for submitting messages. It has approximately the functionality normally assumed by */bin/mail*, except that reading mail is not supported, only submission. That is, it takes a list of addresses on the command line and reads the body of the message on the standard input. By default all the addresses are marked as `To:` recipients. The command has a few options; each option stays in force until overriden.

`-t`: The following arguments, up to the next flag, are `To:` addresses.

`-c`: The following arguments are `Cc:` addresses.

`-s`: The following arguments are text for the `Subject:` line.

This program is most useful in scripts and programs as a simple way of sending a message.

3.2 Message Status Checkup

PP provides a tool, *checkmail*, that connects to an MTA and finds out about messages related to the user that are currently queued.

There are two flags to alter *checkmail*'s behaviour.

-a: This flag prompts *checkmail* to find out about messages to and from the user. By default, *checkmail* only finds out about messages from the user.

-h MTAname: This flag prompts *checkmail* to connect to the named MTA. By default, *checkmail* attempts to connect to an MTA running on the local machine.

Here are two example displays from the *checkmail* tool:

```
john% checkmail -h sheriff
Please wait while attempt to connect to sheriff....connected
The messages from pac on sheriff are as follows:
msg.aa03130 is pending delivery to p.pat@cs.nott.ac.uk

john% checkmail -h sheriff -a
Please wait while attempt to connect to sheriff....connected
The messages to or from pac on sheriff are as follows:
msg.aa03130 is pending delivery to p.pat@cs.nott.ac.uk
msg.aa03140 is from  P.Pat@cs.nott.ac.uk
```

3.3 FTBP Utilities

NOTE: The interfaces described herein will be substantially changed in a future release.

PP provides two utilities for sending and receiving files by mail using the X.400 File Transfer Body Part (FTBP): *fsn*, the PP file transfer sending program; and *fex* the file transfer extraction program.

To try out the FTBP utilities, try the command

```
fsn -rfc822 'whoami' meddle.txt
```

where *meddle.txt* is a text file in the current directory.

Fsn should respond with something like:

```
Processing "meddle.txt" ...
File sent using garfield.isode.com as outbound MTA.
The recipient(s):
   1) RFC822 -> "J.Haxby@garfield.isode.com"
      X400   -> "/I=J/S=Haxby/ADMD=IC/C=WS/"
```

If you get an error message, FTBP is probably not installed on your system. Contact your system administrator for further help.

After a few minutes you should get a message telling you that you have received a file and explaining how to extract it from your FTBP spooling area. If the mail message actually contains the file in some encoded form, then your system administrator may not have set up an FTBP spooling area for you. However, assuming that the file *has* been transferred to your FTBP spooling area, try the command

```
fex -n meddle.tex
```

fex will respond with a message something like:

```
Extracting
Message : msg.13217-0
File    : meddle.txt
Size    : 3612
Wrap    :
Sender  : J.Haxby@odie.garfield.isode.com.
fex: "meddle.txt" exists. Overwrite?(y/n)
```

The file exists because you've just sent it, and presumably, you're still in the same directory you sent it from. At this stage, you probably don't want to overwrite the original file, so respond with `n` and give a different file name.

The next two sections describe the *fex* and *fsn* programs in more detail.

3.3.1 *fsn* – file transfer sending program

The *fsn* program takes a series of options describing how to process the file(s), the addresses of the people you want to send the file(s) to, and, finally, a list of files to be sent, one in each mail message.

The options are:

`-altname` *filename* The recipient will receive the file under this name and not the original name of the file.

`-applref` *string* Descriptive application reference. The *string* describes the application program that the file being sent "belongs" to.

`-apploid` *oid* Registered application reference OID. The *oid* is the registered OID for the application that the file "belongs" to.

`-binary` Forces the type of file being sent to be of binary type, regardless of what it actually is. This might lead to the file being incorrectly extracted by the recipient.

`-comment` *fsn* will prompt for a description of the file to be sent to the recipient. The description may be no more than 400 characters.

`-keepwrap` *wrap filename* Process *filename* according *wrap* before sending it, this might, for example, create a compressed *tar* archive from a directory to be sent to the recipient.

`-wrap` *wrap filename* Same as `-keepwrap` except that `filename` is removed after being sent.

`-listwraps` List the wrappings supported by `-keepwrap` and `-wrap`, no message is sent.

`-output` *filename* Save the temporary file created by *fsn* in *filename*.

`-rfc822` *address* The recipient's address in RFC 822 (Internet) format. The file is sent to this address. Multiple recipients may be specified with repeated `-rfc822` and `-x400` flags.

`-text` Force the type of the file to be text, regardless of what it actually is. This might lead to the file being incorrectly extracted by the recipient. Note that binary files should never be transmitted as text.

`-x400` *address* The recipient's address in X.400 format. The file is sent to this address. Multiple recipients may be specified with repeated `-rfc822` and `-x400` flags.

3.3.2 *fex* – file transfer extraction program

The *fex* program extracts files from the FTBP spool area. It takes a list of options and, optionally, a list of message-ids.

The options are:

`-donotunwrap` Tells *fex* not to run the unwrapping commands on the extracted file.

`-extlist` Give a long listing of the files waiting in the FTBP spool area.

`-file` *filename* Extract from the encoded *filename* instead of a file in the spool area.

`-index` *indices*... The files indicated by *indices* (as produced by `-extlist` or `-list`) are extracted. This must be the last option specified.

`-junk` Discard the specified files.

`-list` Give a brief listing of all the files available for extraction.

`-name` *names*... Extract the files in the spool area named by *names*. The names may include the wildcard `*`, although care must be taken not to conflict with the shell's wildcard expansion (if necessary, the arguments should be quoted). This must be the last option specified.

`-output` *filename* Extract the specified file to *filename*. Normally, a file is extracted to the current directory using the name under which it was sent, but this option allows a different file to be specified. If several files are being extracted, this option only applies to the first one.

`-unwrap` *wrap* Unwrap the file using the specified *wrap*. Normally, this is not done as *fex* will try to decide for itself whether a file needs unwrapping.

`-Info` Display information about a file. Normally used in conjunction with the `-file` option.

`-Path` Extract files to the full path specification in the body part. Normally *fex* extracts to the current directory instead.