exact matches including white space with the exception of letter case not being significant. These variables are distinguished by always starting with a dollar symbol ("$") followed by the field name in parentheses. (e.g., `$(to)` or `$(from)`).

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

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 thedelivery 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.

2.1.2 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:

```
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 2.1: Example .mailfilter File

```
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 2.1.
The grammar is specified in Figure 2.2 on page 16.

2.2 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%
```

2.3 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

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

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 bares 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.

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 new lines removed and multiple white space compressed to single spaces.

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 2.1 on page 8 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 of *.alert* in the home directory.

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

xalert has only some command line switches. These are:

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

`-user username:` *xalert* filters out 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 if 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.

2.3.1 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 *.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. There are three buttons:

`Quit:` which does the obvious thing.

`Clear:` which clears the message window.

`Options:` which pops up a menu with a number of sub-options. These are:

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

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

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

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

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.

2.3.2 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
```

2.4 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 user, even if the message is not sent directly to this user.

`-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 *ttripnote*).

`-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.

2.5 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).

2.6 Delivery Tools

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

2.6.1 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
```

2.6.2 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> "}"                                  10

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

<expression> ::=  "(" <expression> ")"
                  | <expression> "==" <regex>
                  | <expression> "!=" <regex>
                  | <variable> "=" <expression>
                  | <expression> "==" <expression>
                  | <expression> "!=" <expression>
                  | <expression> "||" <expression>                           20
                  | <expression> "&&" <expression>
                  | "!" <expression>
                  | [ "ignore" ] "file" <string>
                  | [ "ignore" ] "pipe" <string>
                  | "exit" [ <expression> ]
                  | <field>
                  | <variable>
                  | <string>

<prexp> ::= <expression>                                                     30
                  | <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 2.2: Grammar for the .mailfilter File