Exim's interfaces to mail filtering

1. Forwarding and filtering in Exim **1**

1.1 Introduction 1
1.2 Filter operation 1
1.3 Testing a new filter file 1
1.4 Installing a filter file 2
1.5 Testing an installed filter file 3
1.6 Details of filtering commands 3

2. Sieve filter files **4**

2.1 Recognition of Sieve filters 4
2.2 Saving to specified folders 4
2.3 Strings containing header names 4
2.4 Exists test with empty list of headers 4
2.5 Header test with invalid MIME encoding in header 4
2.6 Address test for multiple addresses per header 5
2.7 Semantics of keep 5
2.8 Semantics of fileinto 5
2.9 Semantics of redirect 5
2.10 String arguments 5
2.11 Number units 5
2.12 RFC compliance 5

3. Exim filter files **7**

3.1 Format of Exim filter files 7
3.2 Data values in filter commands 7
3.3 String expansion 7
3.4 Some useful general variables 8
3.5 Header variables 9
3.6 User variables 10
3.7 Current directory 10
3.8 Significant deliveries 10
3.9 Filter commands 10
3.10 The add command 11
3.11 The deliver command 11
3.12 The save command 11
3.13 The pipe command 12
3.14 Mail commands 14
3.15 Logging commands 16
3.16 The finish command 16
3.17 The testprint command 16
3.18 The fail command 16
3.19 The freeze command 17
3.20 The headers command 17
3.21 Obeying commands conditionally 17
3.22 String testing conditions 18
3.23 Numeric testing conditions 19
3.24 Testing for significant deliveries 19
3.25 Testing for error messages 19
3.26 Testing a list of addresses 20
3.27 Testing for personal mail 20
3.28 Alias addresses for the personal condition 21
3.29 Details of the personal condition 21

3.30 Testing delivery status 22
3.31 Multiple personal mailboxes 22
3.32 Ignoring delivery errors 22
3.33 Examples of Exim filter commands 22

1. Forwarding and filtering in Exim

This document describes the user interfaces to Exim's in-built mail filtering facilities, and is copyright © The Exim Maintainers 2026. It corresponds to Exim version 4.100.2.

1.1 Introduction

Most Unix mail transfer agents (programs that deliver mail) permit individual users to specify automatic forwarding of their mail, usually by placing a list of forwarding addresses in a file called *.forward* in their home directories. Exim extends this facility by allowing the forwarding instructions to be a set of rules rather than just a list of addresses, in effect providing "*.forward* with conditions". Operating the set of rules is called *filtering*, and the file that contains them is called a *filter file*.

Exim supports two different kinds of filter file. An *Exim filter* contains instructions in a format that is unique to Exim. A *Sieve filter* contains instructions in the Sieve format that is defined by **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)**. As this is a standard format, Sieve filter files may already be familiar to some users. Sieve files should also be portable between different environments. However, the Exim filtering facility contains more features (such as variable expansion), and better integration with the host environment (such as the use of external processes and pipes).

The choice of which kind of filter to use can be left to the end-user, provided that the system administrator has configured Exim appropriately for both kinds of filter. However, if interoperability is important, Sieve is the only choice.

The ability to use filtering or traditional forwarding has to be enabled by the system administrator, and some of the individual facilities can be separately enabled or disabled. A local document should be provided to describe exactly what has been enabled. In the absence of this, consult your system administrator.

This document describes how to use a filter file and the format of its contents. It is intended for use by end-users. Both Sieve filters and Exim filters are covered. However, for Sieve filters, only issues that relate to the Exim implementation are discussed, since Sieve itself is described elsewhere.

The contents of traditional *.forward* files are not described here. They normally contain just a list of addresses, file names, or pipe commands, separated by commas or newlines, but other types of item are also available. The full details can be found in the chapter on the *redirect* router in the Exim specification, which also describes how the system administrator can set up and control the use of filtering.

1.2 Filter operation

It is important to realize that, in Exim, no deliveries are actually made while a filter or traditional *.forward* file is being processed. Running a filter or processing a traditional *.forward* file sets up future delivery operations, but does not carry them out.

The result of filter or *.forward* file processing is a list of destinations to which a message should be delivered. The deliveries themselves take place later, along with all other deliveries for the message. This means that it is not possible to test for successful deliveries while filtering. It also means that any duplicate addresses that are generated are dropped, because Exim never delivers the same message to the same address more than once.

1.3 Testing a new filter file

Filter files, especially the more complicated ones, should always be tested, as it is easy to make mistakes. Exim provides a facility for preliminary testing of a filter file before installing it. This tests the syntax of the file and its basic operation, and can also be used with traditional *.forward* files.

Because a filter can do tests on the content of messages, a test message is required. Suppose you have a new filter file called *myfilter* and a test message in a file called *test-message*. Assuming that Exim is installed with the conventional path name */usr/sbin/sendmail* (some operating systems use */usr/lib/sendmail*), the following command can be used:

```
/usr/sbin/sendmail -bf myfilter <test-message
```

The **-bf** option tells Exim that the following item on the command line is the name of a filter file that is to be tested. There is also a **-bF** option, which is similar, but which is used for testing system filter files, as opposed to user filter files, and which is therefore of use only to the system administrator.

The test message is supplied on the standard input. If there are no message-dependent tests in the filter, an empty file (*/dev/null*) can be used. A supplied message must start with header lines or the "From " message separator line that is found in many multi-message folder files. Note that blank lines at the start terminate the header lines. A warning is given if no header lines are read.

The result of running this command, provided no errors are detected in the filter file, is a list of the actions that Exim would try to take if presented with the message for real. For example, for an Exim filter, the output

```
Deliver message to: gulliver@lilliput.fict.example
Save message to: /home/lemuel/mail/archive
```

means that one copy of the message would be sent to *gulliver@lilliput.fict.example*, and another would be added to the file */home/lemuel/mail/archive*, if all went well.

The actions themselves are not attempted while testing a filter file in this way; there is no check, for example, that any forwarding addresses are valid. For an Exim filter, if you want to know why a particular action is being taken, add the **-v** option to the command. This causes Exim to output the results of any conditional tests and to indent its output according to the depth of nesting of *if* commands. Further additional output from a filter test can be generated by the *testprint* command, which is described below.

When Exim is outputting a list of the actions it would take, if any text strings are included in the output, non-printing characters therein are converted to escape sequences. In particular, if any text string contains a newline character, this is shown as "\n" in the testing output.

When testing a filter in this way, Exim makes up an "envelope" for the message. The recipient is by default the user running the command, and so is the sender, but the command can be run with the **-f** option to supply a different sender. For example,

```
/usr/sbin/sendmail -bf myfilter \
   -f islington@never.where <test-message
```

Alternatively, if the **-f** option is not used, but the first line of the supplied message is a "From " separator from a message folder file (not the same thing as a *From:* header line), the sender is taken from there. If **-f** is present, the contents of any "From " line are ignored.

The "return path" is the same as the envelope sender, unless the message contains a *Return-path:* header, in which case it is taken from there. You need not worry about any of this unless you want to test out features of a filter file that rely on the sender address or the return path.

It is possible to change the envelope recipient by specifying further options. The **-bfd** option changes the domain of the recipient address, while the **-bfl** option changes the "local part", that is, the part before the @ sign. An adviser could make use of these to test someone else's filter file.

The **-bfp** and **-bfs** options specify the prefix or suffix for the local part. These are relevant only when support for multiple personal mailboxes is implemented; see the description in section 3.31 below.

1.4 Installing a filter file

A filter file is normally installed under the name *.forward* in your home directory – it is distinguished from a conventional *.forward* file by its first line (described below). However, the file name is configurable, and some system administrators may choose to use some different name or location for filter files.

1.5 Testing an installed filter file

Testing a filter file before installation cannot find every potential problem; for example, it does not actually run commands to which messages are piped. Some "live" tests should therefore also be done once a filter is installed.

If at all possible, test your filter file by sending messages from some other account. If you send a message to yourself from the filtered account, and delivery fails, the error message will be sent back to the same account, which may cause another delivery failure. It won't cause an infinite sequence of such messages, because delivery failure messages do not themselves generate further messages. However, it does mean that the failure won't be returned to you, and also that the postmaster will have to investigate the stuck message.

If you have to test an Exim filter from the same account, a sensible precaution is to include the line

```
if error_message then finish endif
```

as the first filter command, at least while testing. This causes filtering to be abandoned for a delivery failure message, and since no destinations are generated, the message goes on to be delivered to the original address. Unless there is a good reason for not doing so, it is recommended that the above test be left in all Exim filter files. (This does not apply to Sieve files.)

1.6 Details of filtering commands

The filtering commands for Sieve and Exim filters are completely different in syntax and semantics. The Sieve mechanism is defined in **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)**; in the next chapter we describe how it is integrated into Exim. The subsequent chapter covers Exim filtering commands in detail.

2. Sieve filter files

The code for Sieve filtering in Exim was contributed by Michael Haardt, and most of the content of this chapter is taken from the notes he provided. Since Sieve is an extensible language, it is important to understand "Sieve" in this context as "the specific implementation of Sieve for Exim".

This chapter does not contain a description of Sieve, since that can be found in **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)**, which should be read in conjunction with these notes.

The Exim Sieve implementation offers the core as defined by the RFC, comparison tests, the subaddress parameter, the **copy**, **envelope**, **fileinto**, **notify**, and **vacation** extensions, but not the **reject** extension. Exim did not support message delivery notifications (MDNs) when Sieve handling was introduced, so **reject** (which needs them) could not be supported.

Additionally, if Exim is built with iconv() supoprt and WITH_CONTENT_SCAN defined, the implementation supports the **body** extension defined by **RFC 5173 (https://datatracker.ietf.org/doc/html/rfc5173)**.

In order for Sieve to work properly in Exim, the system administrator needs to make some adjustments to the Exim configuration. These are described in the chapter on the *redirect* router in the full Exim specification.

2.1 Recognition of Sieve filters

A filter file is interpreted as a Sieve filter if its first line is

```
# Sieve filter
```

This is what distinguishes it from a conventional *.forward* file or an Exim filter file.

2.2 Saving to specified folders

If the system administrator has set things up as suggested in the Exim specification, and you use *keep* or *fileinto* to save a mail into a folder, absolute files are stored where specified, relative files are stored relative to *$home*, and *inbox* goes to the standard mailbox location.

2.3 Strings containing header names

RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028) does not specify what happens if a string denoting a header field does not contain a valid header name, for example, it contains a colon. This implementation generates an error instead of ignoring the header field in order to ease script debugging, which fits in with the common picture of Sieve.

2.4 Exists test with empty list of headers

The **exists** test succeeds only if all the specified headers exist. **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)** does not explicitly specify what happens on an empty list of headers. This implementation evaluates that condition as true, interpreting the RFC in a strict sense.

2.5 Header test with invalid MIME encoding in header

Some MUAs process invalid base64 encoded data, generating junk. Others ignore junk after seeing an equal sign in base64 encoded data. **RFC 2047 (https://datatracker.ietf.org/doc/html/rfc2047)** does not specify how to react in this case, other than stating that a client must not forbid to process a message for that reason. **RFC 2045 (https://datatracker.ietf.org/doc/html/rfc2045)** specifies that invalid data should be ignored (apparently looking at end of line characters). It also specifies that invalid data may lead to rejecting messages containing them (and there it appears to talk about true encoding violations), which is a clear contradiction to ignoring them.

RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028) does not specify how to process incorrect MIME words. This implementation treats them literally, as it does if the word is correct but its character set cannot be converted to UTF-8.

2.6 Address test for multiple addresses per header

A header may contain multiple addresses. **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)** does not explicitly specify how to deal with them, but since the address test checks if anything matches anything else, matching one address suffices to satisfy the condition. That makes it impossible to test if a header contains a certain set of addresses and no more, but it is more logical than letting the test fail if the header contains an additional address besides the one the test checks for.

2.7 Semantics of keep

The *keep* command is by default equivalent to

```
fileinto "inbox";
```

but, when called from a redirect router, the destination can be set by using the **sieve_inbox** option on the router. It saves the message and resets the implicit keep flag. It does not set the implicit keep flag; there is no command to set it once it has been reset.

2.8 Semantics of fileinto

RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028) does not specify whether *fileinto* should try to create a mail folder if it does not exist. This implementation allows the sysadmin to configure that aspect using the *appendfile* transport options **create_directory**, **create_file**, and **file_must_exist**. See the *appendfile* transport in the Exim specification for details.

2.9 Semantics of redirect

Sieve scripts are supposed to be interoperable between servers, so this implementation does not allow mail to be redirected to unqualified addresses, because the domain would depend on the system being used. On systems with virtual mail domains, the default domain is probably not what the user expects it to be.

2.10 String arguments

There has been confusion if the string arguments to *require* are to be matched case-sensitively or not. This implementation matches them with the match type *:is* (default, see section 2.7.1 of **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)**) and the comparator *i;ascii-casemap* (default, see section 2.7.3 of the RFC). The RFC defines the command defaults clearly, so any different implementations are in violation. The same is valid for comparator names, also specified as strings.

2.11 Number units

There is a mistake in **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)**: the suffix G denotes gibi-, not tebibyte. The mistake is obvious, because the RFC specifies G to denote 2^30 (which is gibi, not tebi), and that is what this implementation uses as the scaling factor for the suffix G.

2.12 RFC compliance

Exim requires the first line of a Sieve filter to be

```
# Sieve filter
```

Of course the RFC does not specify that line. Do not expect examples to work without adding it, though.

RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028) requires the use of CRLF to terminate a line. The rationale was that CRLF is universally used in network protocols to mark the end of the line. This implementation does not embed Sieve in a network protocol, but uses Sieve scripts as part of the Exim MTA. Since all parts of Exim use LF as the newline character, this implementation does, too, by default, though the system administrator may choose (at Exim compile time) to use CRLF instead.

Exim violates **RFC 2822 (https://datatracker.ietf.org/doc/html/rfc2822)**, section 3.6.8, by accepting 8-bit header names, so this implementation repeats this violation to stay consistent with Exim. This is in preparation for UTF-8 data.

Sieve scripts cannot contain NUL characters in strings, but mail headers could contain MIME encoded NUL characters, which could never be matched by Sieve scripts using exact comparisons. For that reason, this implementation extends the Sieve quoted string syntax with \0 to describe a NUL character, violating \0 being the same as 0 in **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)**. Even without using \0, the following tests are all true in this implementation. Implementations that use C-style strings will only evaluate the first test as true.

```
Subject: =?iso-8859-1?q?abc=00def

header :contains "Subject" ["abc"]
header :contains "Subject" ["def"]
header :matches "Subject" ["abc?def"]
```

Note that by considering Sieve to be an MUA, **RFC 2047 (https://datatracker.ietf.org/doc/html/rfc2047)** can be interpreted in a way that NUL characters truncating strings is allowed for Sieve implementations, although not recommended. It is further allowed to use encoded NUL characters in headers, but that's not recommended either. The above example shows why.

RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028) states that if an implementation fails to convert a character set to UTF-8, two strings cannot be equal if one contains octets greater than 127. Assuming that all unknown character sets are one-byte character sets with the lower 128 octets being US-ASCII is not sound, so this implementation violates **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)** and treats such MIME words literally. That way at least something could be matched.

The folder specified by *fileinto* must not contain the character sequence ".." to avoid security problems. **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)** does not specify the syntax of folders apart from *keep* being equivalent to

```
fileinto "INBOX";
```

This implementation uses *inbox* instead.

Sieve script errors currently cause messages to be silently filed into *inbox*. **RFC 3028 (https://datatracker.ietf.org/doc/html/rfc3028)** requires that the user is notified of that condition. This may be implemented in the future by adding a header line to mails that are filed into *inbox* due to an error in the filter.

3. Exim filter files

This chapter contains a full description of the contents of Exim filter files.

3.1 Format of Exim filter files

Apart from leading white space, the first text in an Exim filter file must be

```
# Exim filter
```

This is what distinguishes it from a conventional *.forward* file or a Sieve filter file. If the file does not have this initial line (or the equivalent for a Sieve filter), it is treated as a conventional *.forward* file, both when delivering mail and when using the **-bf** testing mechanism. The white space in the line is optional, and any capitalization may be used. Further text on the same line is treated as a comment. For example, you could have

```
#   Exim filter   <<== do not edit or remove this line!
```

The remainder of the file is a sequence of filtering commands, which consist of keywords and data values. For example, in the command

```
deliver gulliver@lilliput.fict.example
```

the keyword is `deliver` and the data value is `gulliver@lilliput.fict.example`. White space or line breaks separate the components of a command, except in the case of conditions for the *if* command, where round brackets (parentheses) also act as separators. Complete commands are separated from each other by white space or line breaks; there are no special terminators. Thus, several commands may appear on one line, or one command may be spread over a number of lines.

If the character # follows a separator anywhere in a command, everything from # up to the next newline is ignored. This provides a way of including comments in a filter file.

3.2 Data values in filter commands

There are two ways in which a data value can be input:

- If the text contains no white space, it can be typed verbatim. However, if it is part of a condition, it must also be free of round brackets (parentheses), as these are used for grouping in conditions.
- Otherwise, text must be enclosed in double quotation marks. In this case, the character \ (backslash) is treated as an "escape character" within the string, causing the following character or characters to be treated specially:

  `\n` is replaced by a newline
  `\r` is replaced by a carriage return
  `\t` is replaced by a tab

Backslash followed by up to three octal digits is replaced by the character specified by those digits, and `\x` followed by up to two hexadecimal digits is treated similarly. Backslash followed by any other character is replaced by the second character, so that in particular, `\"` becomes `"` and `\\` becomes `\`. A data item enclosed in double quotes can be continued onto the next line by ending the first line with a backslash. Any leading white space at the start of the continuation line is ignored.

In addition to the escape character processing that occurs when strings are enclosed in quotes, most data values are also subject to *string expansion* (as described in the next section), in which case the characters `$` and `\` are also significant. This means that if a single backslash is actually required in such a string, and the string is also quoted, `\\\\` has to be entered.

The maximum permitted length of a data string, before expansion, is 1024 characters.

3.3 String expansion

Most data values are expanded before use. Expansion consists of replacing substrings beginning with `$` with other text. The full expansion facilities available in Exim are extensive. If you want to know

everything that Exim can do with strings, you should consult the chapter on string expansion in the Exim documentation.

In filter files, by far the most common use of string expansion is the substitution of the contents of a variable. For example, the substring

```
$reply_address
```

is replaced by the address to which replies to the message should be sent. If such a variable name is followed by a letter or digit or underscore, it must be enclosed in curly brackets (braces), for example,

```
${reply_address}
```

If a `$` character is actually required in an expanded string, it must be escaped with a backslash, and because backslash is also an escape character in quoted input strings, it must be doubled in that case. The following two examples illustrate two different ways of testing for a `$` character in a message:

```
if $message_body contains \$ then ...
if $message_body contains "\\$" then ...
```

You can prevent part of a string from being expanded by enclosing it between two occurrences of `\N`. For example,

```
if $message_body contains \N$$$$\N then ...
```

tests for a run of four dollar characters.

3.4 Some useful general variables

A complete list of the available variables is given in the Exim documentation. This shortened list contains the ones that are most likely to be useful in personal filter files:

$body_linecount: The number of lines in the body of the message.

$body_zerocount: The number of binary zero characters in the body of the message.

$home: In conventional configurations, this variable normally contains the user's home directory. The system administrator can, however, change this.

$local_part: The part of the email address that precedes the @ sign – normally the user's login name. If support for multiple personal mailboxes is enabled (see section 3.31 below) and a prefix or suffix for the local part was recognized, it is removed from the string in this variable.

$local_part_prefix: If support for multiple personal mailboxes is enabled (see section 3.31 below), and a local part prefix was recognized, this variable contains the prefix. Otherwise it contains an empty string.

$local_part_suffix: If support for multiple personal mailboxes is enabled (see section 3.31 below), and a local part suffix was recognized, this variable contains the suffix. Otherwise it contains an empty string.

$message_body: The initial portion of the body of the message. By default, up to 500 characters are read into this variable, but the system administrator can configure this to some other value. Newlines in the body are converted into single spaces.

$message_body_end: The final portion of the body of the message, formatted and limited in the same way as *$message_body*.

$message_body_size: The size of the body of the message, in bytes.

$message_exim_id: The message's local identification string, which is unique for each message handled by a single host.

$message_headers: The header lines of the message, concatenated into a single string, with newline characters between them.

$message_size: The size of the entire message, in bytes.

$original_local_part: When an address that arrived with the message is being processed, this contains the same value as the variable *$local_part*. However, if an address generated by an alias, forward, or filter file is being processed, this variable contains the local part of the original address.

$reply_address: The contents of the *Reply-to:* header, if the message has one; otherwise the contents of the *From:* header. It is the address to which normal replies to the message should be sent.

$return_path: The return path – that is, the sender field that will be transmitted as part of the message's envelope if the message is sent to another host. This is the address to which delivery errors are sent. In many cases, this variable has the same value as *$sender_address*, but if, for example, an incoming message to a mailing list has been expanded, *$return_path* may have been changed to contain the address of the list maintainer.

$sender_address: The sender address that was received in the envelope of the message. This is not necessarily the same as the contents of the *From:* or *Sender:* header lines. For delivery error messages ("bounce messages") there is no sender address, and this variable is empty.

$tod_full: A full version of the time and date, for example: Wed, 18 Oct 1995 09:51:40 +0100. The timezone is always given as a numerical offset from GMT.

$tod_log: The time and date in the format used for writing Exim's log files, without the timezone, for example: 1995-10-12 15:32:29.

$tod_zone: The local timezone offset, for example: +0100.

3.5 Header variables

There is a special set of expansion variables containing the header lines of the message being processed. These variables have names beginning with *$header_* followed by the name of the header line, terminated by a colon. For example,

```
$header_from:
$header_subject:
```

The whole item, including the terminating colon, is replaced by the contents of the message header line. If there is more than one header line with the same name, their contents are concatenated. For header lines whose data consists of a list of addresses (for example, *From:* and *To:*), a comma and newline is inserted between each set of data. For all other header lines, just a newline is used.

Leading and trailing white space is removed from header line data, and if there are any MIME "words" that are encoded as defined by **RFC 2047 (https://datatracker.ietf.org/doc/html/rfc2047)** (because they contain non-ASCII characters), they are decoded and translated, if possible, to a local character set. Translation is attempted only on operating systems that have the *iconv()* function. This makes the header line look the same as it would when displayed by an MUA. The default character set is ISO-8859-1, but this can be changed by means of the *headers* command (see below).

If you want to see the actual characters that make up a header line, you can specify *$rheader_* instead of *$header_*. This inserts the "raw" header line, unmodified.

There is also an intermediate form, requested by *$bheader_*, which removes leading and trailing space and decodes MIME "words", but does not do any character translation. If an attempt to decode what looks superficially like a MIME "word" fails, the raw string is returned. If decoding produces a binary zero character, it is replaced by a question mark.

The capitalization of the name following *$header_* is not significant. Because any printing character except colon may appear in the name of a message's header (this is a requirement of **RFC 2822 (https://datatracker.ietf.org/doc/html/rfc2822)**, the document that describes the format of a mail message) curly brackets must *not* be used in this case, as they will be taken as part of the header name. Two shortcuts are allowed in naming header variables:

- The initiating *$header_*, *$rheader_*, or *$bheader_* can be abbreviated to *$h_*, *$rh_*, or *$bh_*, respectively.

- The terminating colon can be omitted if the next character is white space. The white space character is retained in the expanded string. However, this is not recommended, because it makes it easy to forget the colon when it really is needed.

If the message does not contain a header of the given name, an empty string is substituted. Thus it is important to spell the names of headers correctly. Do not use *$header_Reply_to* when you really mean *$header_Reply-to*.

3.6 User variables

There are ten user variables with names *$n0 – $n9* that can be incremented by the *add* command (see section 3.10). These can be used for "scoring" messages in various ways. If Exim is configured to run a "system filter" on every message, the values left in these variables are copied into the variables *$sn0 – $sn9* at the end of the system filter, thus making them available to users' filter files. How these values are used is entirely up to the individual installation.

3.7 Current directory

The contents of your filter file should not make any assumptions about the current directory. It is best to use absolute paths for file names; you can normally make use of the *$home* variable to refer to your home directory. The *save* command automatically inserts *$home* at the start of non-absolute paths.

3.8 Significant deliveries

When in the course of delivery a message is processed by a filter file, what happens next, that is, after the filter file has been processed, depends on whether or not the filter sets up any *significant deliveries*. If at least one significant delivery is set up, the filter is considered to have handled the entire delivery arrangements for the current address, and no further processing of the address takes place. If, however, no significant deliveries are set up, Exim continues processing the current address as if there were no filter file, and typically sets up a delivery of a copy of the message into a local mailbox. In particular, this happens in the special case of a filter file containing only comments.

The delivery commands *deliver*, *save*, and *pipe* are by default significant. However, if such a command is preceded by the word "unseen", its delivery is not considered to be significant. In contrast, other commands such as *mail* and *vacation* do not set up significant deliveries unless preceded by the word "seen". The following example commands set up significant deliveries:

```
deliver jack@beanstalk.example
pipe $home/bin/mymailscript
seen mail subject "message discarded"
seen finish
```

The following example commands do not set up significant deliveries:

```
unseen deliver jack@beanstalk.example
unseen pipe $home/bin/mymailscript
mail subject "message discarded"
finish
```

3.9 Filter commands

The filter commands that are described in subsequent sections are listed below, with the section in which they are described in brackets:

add	increment a user variable (section 3.10)
deliver	deliver to an email address (section 3.11)
fail	force delivery failure (sysadmin use) (section 3.18)
finish	end processing (section 3.16)
freeze	freeze message (sysadmin use) (section 3.19)
headers	set the header character set (section 3.20)
if	test condition(s) (section 3.21)

logfile	define log file (section 3.15)
logwrite	write to log file (section 3.15)
mail	send a reply message (section 3.14)
pipe	pipe to a command (section 3.13)
save	save to a file (section 3.12)
testprint	print while testing (section 3.17)
vacation	tailored form of *mail* (section 3.14)

The *headers* command has additional parameters that can be used only in a system filter. The *fail* and *freeze* commands are available only when Exim's filtering facilities are being used as a system filter, and are therefore usable only by the system administrator and not by ordinary users. They are mentioned only briefly in this document; for more information, see the main Exim specification.

3.10 The add command

```
      add <number> to <user variable>
e.g. add 2 to n3
```

There are 10 user variables of this type, with names *$n0 – $n9*. Their values can be obtained by the normal expansion syntax (for example *$n3*) in other commands. At the start of filtering, these variables all contain zero. Both arguments of the *add* command are expanded before use, making it possible to add variables to each other. Subtraction can be obtained by adding negative numbers.

3.11 The deliver command

```
      deliver <mail address>
e.g. deliver "Dr Livingstone <David@somewhere.africa.example>"
```

This command provides a forwarding operation. The delivery that it sets up is significant unless the command is preceded by "unseen" (see section 3.8). The message is sent on to the given address, exactly as happens if the address had appeared in a traditional *.forward* file. If you want to deliver the message to a number of different addresses, you can use more than one *deliver* command (each one may have only one address). However, duplicate addresses are discarded.

To deliver a copy of the message to your normal mailbox, your login name can be given as the address. Once an address has been processed by the filtering mechanism, an identical generated address will not be so processed again, so doing this does not cause a loop.

However, if you have a mail alias, you should *not* refer to it here. For example, if the mail address *L.Gulliver* is aliased to *lg303* then all references in Gulliver's *.forward* file should be to *lg303*. A reference to the alias will not work for messages that are addressed to that alias, since, like *.forward* file processing, aliasing is performed only once on an address, in order to avoid looping.

Following the new address, an optional second address, preceded by "errors_to" may appear. This changes the address to which delivery errors on the forwarded message will be sent. Instead of going to the message's original sender, they go to this new address. For ordinary users, the only value that is permitted for this address is the user whose filter file is being processed. For example, the user *lg303* whose mailbox is in the domain *lilliput.example* could have a filter file that contains

```
deliver jon@elsewhere.example errors_to lg303@lilliput.example
```

Clearly, using this feature makes sense only in situations where not all messages are being forwarded. In particular, bounce messages must not be forwarded in this way, as this is likely to create a mail loop if something goes wrong.

3.12 The save command

```
      save <file name>
e.g. save $home/mail/bookfolder
```

This command specifies that a copy of the message is to be appended to the given file (that is, the file is to be used as a mail folder). The delivery that *save* sets up is significant unless the command is preceded by "unseen" (see section 3.8).

More than one *save* command may be obeyed; each one causes a copy of the message to be written to its argument file, provided they are different (duplicate *save* commands are ignored).

If the file name does not start with a / character, the contents of the *$home* variable are prepended, unless it is empty, or the system administrator has disabled this feature. In conventional configurations, this variable is normally set in a user filter to the user's home directory, but the system administrator may set it to some other path. In some configurations, *$home* may be unset, or prepending may be disabled, in which case a non-absolute path name may be generated. Such configurations convert this to an absolute path when the delivery takes place. In a system filter, *$home* is never set.

The user must of course have permission to write to the file, and the writing of the file takes place in a process that is running as the user, under the user's primary group. Any secondary groups to which the user may belong are not normally taken into account, though the system administrator can configure Exim to set them up. In addition, the ability to use this command at all is controlled by the system administrator – it may be forbidden on some systems.

An optional mode value may be given after the file name. The value for the mode is interpreted as an octal number, even if it does not begin with a zero. For example:

```
save /some/folder 640
```

This makes it possible for users to override the system-wide mode setting for file deliveries, which is normally 600. If an existing file does not have the correct mode, it is changed.

An alternative form of delivery may be enabled on your system, in which each message is delivered into a new file in a given directory. If this is the case, this functionality can be requested by giving the directory name terminated by a slash after the *save* command, for example

```
save separated/messages/
```

There are several different formats for such deliveries; check with your system administrator or local documentation to find out which (if any) are available on your system. If this functionality is not enabled, the use of a path name ending in a slash causes an error.

3.13 The pipe command

```
      pipe <command>
e.g. pipe "$home/bin/countmail $sender_address"
```

This command specifies that the message is to be delivered to the specified command using a pipe. The delivery that it sets up is significant unless the command is preceded by "unseen" (see section 3.8). Remember, however, that no deliveries are done while the filter is being processed. All deliveries happen later on. Therefore, the result of running the pipe is not available to the filter.

When the deliveries are done, a separate process is run, and a copy of the message is passed on its standard input. The process runs as the user, under the user's primary group. Any secondary groups to which the user may belong are not normally taken into account, though the system administrator can configure Exim to set them up. More than one *pipe* command may appear; each one causes a copy of the message to be written to its argument pipe, provided they are different (duplicate *pipe* commands are ignored).

When the time comes to transport the message, the command supplied to *pipe* is split up by Exim into a command name and a number of arguments. These are delimited by white space except for arguments enclosed in double quotes, in which case backslash is interpreted as an escape, or in single quotes, in which case no escaping is recognized. Note that as the whole command is normally supplied in double quotes, a second level of quoting is required for internal double quotes. For example:

```
pipe "$home/myscript \"size is $message_size\""
```

String expansion is performed on the separate components after the line has been split up, and the command is then run directly by Exim; it is not run under a shell. Therefore, substitution cannot change the number of arguments, nor can quotes, backslashes or other shell metacharacters in variables cause confusion.

Documentation for some programs that are normally run via this kind of pipe often suggest that the command should start with

```
IFS=" "
```

This is a shell command, and should *not* be present in Exim filter files, since it does not normally run the command under a shell.

However, there is an option that the administrator can set to cause a shell to be used. In this case, the entire command is expanded as a single string and passed to the shell for interpretation. It is recommended that this be avoided if at all possible, since it can lead to problems when inserted variables contain shell metacharacters.

The default PATH set up for the command is determined by the system administrator, usually containing at least */bin* and */usr/bin* so that common commands are available without having to specify an absolute file name. However, it is possible for the system administrator to restrict the pipe facility so that the command name must not contain any / characters, and must be found in one of the directories in the configured PATH. It is also possible for the system administrator to lock out the use of the *pipe* command altogether.

When the command is run, a number of environment variables are set up. The complete list for pipe deliveries may be found in the Exim reference manual. Those that may be useful for pipe deliveries from user filter files are:

DOMAIN	the domain of the address
HOME	your home directory
LOCAL_PART	see below
LOCAL_PART_PREFIX	see below
LOCAL_PART_SUFFIX	see below
LOGNAME	your login name
MESSAGE_ID	the unique id of the message
PATH	the command search path
RECIPIENT	the complete recipient address
SENDER	the sender of the message
SHELL	/bin/sh
USER	see below

LOCAL_PART, LOGNAME, and USER are all set to the same value, namely, your login id. LOCAL_PART_PREFIX and LOCAL_PART_SUFFIX may be set if Exim is configured to recognize prefixes or suffixes in the local parts of addresses. For example, a message addressed to *pat-suf2@domain.example* may cause the filter for user *pat* to be run. If this sets up a pipe delivery, LOCAL_PART_SUFFIX is `-suf2` when the pipe command runs. The system administrator has to configure Exim specially for this feature to be available.

If you run a command that is a shell script, be very careful in your use of data from the incoming message in the commands in your script. **RFC 2822 (https://datatracker.ietf.org/doc/html/rfc2822)** is very generous in the characters that are permitted to appear in mail addresses, and in particular, an address may begin with a vertical bar or a slash. For this reason you should always use quotes round any arguments that involve data from the message, like this:

```
/some/command '$SENDER'
```

so that inserted shell meta-characters do not cause unwanted effects.

Remember that, as was explained earlier, the pipe command is not run at the time the filter file is interpreted. The filter just defines what deliveries are required for one particular addressee of a message. The deliveries themselves happen later, once Exim has decided everything that needs to be done for the message.

A consequence of this is that you cannot inspect the return code from the pipe command from within the filter. Nevertheless, the code returned by the command is important, because Exim uses it to decide whether the delivery has succeeded or failed.

The command should return a zero completion code if all has gone well. Most non-zero codes are treated by Exim as indicating a failure of the pipe. This is treated as a delivery failure, causing the message to be returned to its sender. However, there are some completion codes that are treated as temporary errors. The message remains on Exim's spool disk, and the delivery is tried again later, though it will ultimately time out if the delivery failures go on too long. The completion codes to which this applies can be specified by the system administrator; the default values are 73 and 75.

The pipe command should not normally write anything to its standard output or standard error file descriptors. If it does, whatever is written is normally returned to the sender of the message as a delivery error, though this action can be varied by the system administrator.

3.14 Mail commands

There are two commands that cause the creation of a new mail message, neither of which count as a significant delivery unless the command is preceded by the word "seen" (see section 3.8). This is a powerful facility, but it should be used with care, because of the danger of creating infinite sequences of messages. The system administrator can forbid the use of these commands altogether.

To help prevent runaway message sequences, these commands have no effect when the incoming message is a bounce (delivery error) message, and messages sent by this means are treated as if they were reporting delivery errors. Thus, they should never themselves cause a bounce message to be returned. The basic mail-sending command is

```
mail [to <address-list>]
     [cc <address-list>]
     [bcc <address-list>]
     [from <address>]
     [reply_to <address>]
     [subject <text>]
     [extra_headers <text>]
     [text <text>]
     [[expand] file <filename>]
     [return message]
     [log <log file name>]
     [once <note file name>]
     [once_repeat <time interval>]
e.g. mail text "Your message about $h_subject: has been received"
```

Each <*address-list*> can contain a number of addresses, separated by commas, in the format of a *To:* or *Cc:* header line. In fact, the text you supply here is copied exactly into the appropriate header line. It may contain additional information as well as email addresses. For example:

```
mail to "Julius Caesar <jc@rome.example>, \
         <ma@rome.example> (Mark A.)"
```

Similarly, the texts supplied for **from** and **reply_to** are copied into their respective header lines.

If no recipient is specified (by a to, cc or bcc) then the mail is sent to the sender of the message being filtered.

As a convenience for use in one common case, there is also a command called *vacation*. It behaves in the same way as *mail*, except that the defaults for the **subject**, **file**, **log**, **once**, and **once_repeat** options are

```
subject "On vacation"
expand file .vacation.msg
log  .vacation.log
once .vacation
once_repeat 7d
```

respectively. These are the same file names and repeat period used by the traditional Unix *vacation* command. The defaults can be overridden by explicit settings, but if a file name is given its contents are expanded only if explicitly requested.

Warning: The *vacation* command should always be used conditionally, subject to at least the *personal* condition (see section 3.27 below) so as not to send automatic replies to non-personal messages from mailing lists or elsewhere. Sending an automatic response to a mailing list or a mailing list manager is an Internet Sin.

For both commands, the key/value argument pairs can appear in any order. At least one of **text** or **file** must appear (except with *vacation*, where there is a default for **file**); if both are present, the text string appears first in the message. If **expand** precedes **file**, each line of the file is subject to string expansion before it is included in the message.

Several lines of text can be supplied to **text** by including the escape sequence "\n" in the string wherever a newline is required. If the command is output during filter file testing, newlines in the text are shown as "\n".

Note that the keyword for creating a *Reply-To:* header is **reply_to**, because Exim keywords may contain underscores, but not hyphens. If the **from** keyword is present and the given address does not match the user who owns the forward file, Exim normally adds a *Sender:* header to the message, though it can be configured not to do this.

The **extra_headers** keyword allows you to add custom header lines to the message. The text supplied must be one or more syntactically valid **RFC 2822 (https://datatracker.ietf.org/doc/html/rfc2822)** header lines. You can use "\n" within quoted text to specify newlines between headers, and also to define continued header lines. For example:

```
extra_headers "h1: first\nh2: second\n continued\nh3: third"
```

No newline should appear at the end of the final header line.

If no **to** argument appears, the message is sent to the address in the *$reply_address* variable (see section 3.3 above). An *In-Reply-To:* header is automatically included in the created message, giving a reference to the message identification of the incoming message.

If **return message** is specified, the incoming message that caused the filter file to be run is added to the end of the message, subject to a maximum size limitation.

If a log file is specified, a line is added to it for each message sent.

If a **once** file is specified, it is used to hold a database for remembering who has received a message, and no more than one message is ever sent to any particular address, unless **once_repeat** is set. This specifies a time interval after which another copy of the message is sent. The interval is specified as a sequence of numbers, each followed by the initial letter of one of "seconds", "minutes", "hours", "days", or "weeks". For example,

```
once_repeat 5d4h
```

causes a new message to be sent if at least 5 days and 4 hours have elapsed since the last one was sent. There must be no white space in a time interval.

Commonly, the file name specified for **once** is used as the base name for direct-access (DBM) file operations. There are a number of different DBM libraries in existence. Some operating systems provide one as a default, but even in this case a different one may have been used when building Exim. With some DBM libraries, specifying **once** results in two files being created, with the suffixes *.dir* and *.pag* being added to the given name. With some others a single file with the suffix *.db* is used, or the name is used unchanged.

Using a DBM file for implementing the **once** feature means that the file grows as large as necessary. This is not usually a problem, but some system administrators want to put a limit on it. The facility can be configured not to use a DBM file, but instead, to use a regular file with a maximum size. The data in such a file is searched sequentially, and if the file fills up, the oldest entry is deleted to make way for a new one. This means that some correspondents may receive a second copy of the message

after an unpredictable interval. Consult your local information to see if your system is configured this way.

More than one *mail* or *vacation* command may be obeyed in a single filter run; they are all honoured, even when they are to the same recipient.

3.15 Logging commands

A log can be kept of actions taken by a filter file. This facility is normally available in conventional configurations, but there are some situations where it might not be. Also, the system administrator may choose to disable it. Check your local information if in doubt.

Logging takes place while the filter file is being interpreted. It does not queue up for later like the delivery commands. The reason for this is so that a log file need be opened only once for several write operations. There are two commands, neither of which constitutes a significant delivery. The first defines a file to which logging output is subsequently written:

```
      logfile <file name>
e.g. logfile $home/filter.log
```

The file name must be fully qualified. You can use *$home*, as in this example, to refer to your home directory. The file name may optionally be followed by a mode for the file, which is used if the file has to be created. For example,

```
logfile $home/filter.log 0644
```

The number is interpreted as octal, even if it does not begin with a zero. The default for the mode is 600. It is suggested that the *logfile* command normally appear as the first command in a filter file. Once a log file has been obeyed, the *logwrite* command can be used to write to it:

```
      logwrite "<some text string>"
e.g. logwrite "$tod_log $message_id processed"
```

It is possible to have more than one *logfile* command, to specify writing to different log files in different circumstances. A previously opened log is closed on a subsequent *logfile* command. Writing takes place at the end of the file, and a newline character is added to the end of each string if there isn't one already there. Newlines can be put in the middle of the string by using the "\n" escape sequence. Lines from simultaneous deliveries may get interleaved in the file, as there is no interlocking, so you should plan your logging with this in mind. However, data should not get lost.

3.16 The finish command

The command *finish*, which has no arguments, causes Exim to stop interpreting the filter file. This is not a significant action unless preceded by "seen". A filter file containing only "seen finish" is a black hole.

3.17 The testprint command

It is sometimes helpful to be able to print out the values of variables when testing filter files. The command

```
      testprint <text>
e.g. testprint "home=$home reply_address=$reply_address"
```

does nothing when mail is being delivered. However, when the filtering code is being tested by means of the **-bf** option (see section 1.3 above), the value of the string is written to the standard output.

Alternatively, if the "filter" debug channel is enabled (using the **-v** option) standard error is used.

3.18 The fail command

When Exim's filtering facilities are being used as a system filter, the *fail* command is available, to force delivery failure. Because this command is normally usable only by the system administrator,

and not enabled for use by ordinary users, it is described in more detail in the main Exim specification rather than in this document.

3.19 The freeze command

When Exim's filtering facilities are being used as a system filter, the *freeze* command is available, to freeze a message on the queue. Because this command is normally usable only by the system administrator, and not enabled for use by ordinary users, it is described in more detail in the main Exim specification rather than in this document.

3.20 The headers command

The *headers* command can be used to change the target character set that is used when translating the contents of encoded header lines for insertion by the *$header_* mechanism (see section 3.5 above). The default can be set in the Exim configuration; if not specified, ISO-8859-1 is used. The only currently supported format for the *headers* command in user filters is as in this example:

```
headers charset "UTF-8"
```

That is, *headers* is followed by the word "charset" and then the name of a character set. This particular example would be useful if you wanted to compare the contents of a header to a UTF-8 string.

In system filter files, the *headers* command can be used to add or remove header lines from the message. These features are described in the main Exim specification.

3.21 Obeying commands conditionally

Most of the power of filtering comes from the ability to test conditions and obey different commands depending on the outcome. The *if* command is used to specify conditional execution, and its general form is

```
if     <condition>
then   <commands>
elif   <condition>
then   <commands>
else   <commands>
endif
```

There may be any number of *elif* and *then* sections (including none) and the *else* section is also optional. Any number of commands, including nested *if* commands, may appear in any of the <*commands*> sections.

Conditions can be combined by using the words *and* and *or*, and round brackets (parentheses) can be used to specify how several conditions are to combine. Without brackets, *and* is more binding than *or*. For example:

```
if
$h_subject: contains "Make money" or
$h_precedence: is "junk" or
($h_sender: matches ^\\d{8}@ and not personal) or
$message_body contains "this is not spam"
then
seen finish
endif
```

A condition can be preceded by *not* to negate it, and there are also some negative forms of condition that are more English-like.

3.22 String testing conditions

There are a number of conditions that operate on text strings, using the words “begins”, “ends”, “is”, “contains” and “matches”. If you want to apply the same test to more than one header line, you can easily concatenate them into a single string for testing, as in this example:

```
if "$h_to:, $h_cc:" contains me@domain.example then ...
```

If a string-testing condition name is written in lower case, the testing of letters is done without regard to case; if it is written in upper case (for example, “CONTAINS”), the case of letters is taken into account.

```
      <text1> begins <text2>
      <text1> does not begin <text2>
e.g. $header_from: begins "Friend@"
```

A “begins” test checks for the presence of the second string at the start of the first, both strings having been expanded.

```
      <text1> ends <text2>
      <text1> does not end <text2>
e.g. $header_from: ends "public.com.example"
```

An “ends” test checks for the presence of the second string at the end of the first, both strings having been expanded.

```
      <text1> is <text2>
      <text1> is not <text2>
e.g. $local_part_suffix is "-foo"
```

An “is” test does an exact match between the strings, having first expanded both strings.

```
      <text1> contains <text2>
      <text1> does not contain <text2>
e.g. $header_subject: contains "evolution"
```

A “contains” test does a partial string match, having expanded both strings.

```
      <text1> matches <text2>
      <text1> does not match <text2>
e.g. $sender_address matches "(bill|john)@"
```

For a “matches” test, after expansion of both strings, the second one is interpreted as a regular expression. Exim uses the PCRE2 regular expression library, which provides regular expressions that are compatible with Perl.

The match succeeds if the regular expression matches any part of the first string. If you want a regular expression to match only at the start or end of the subject string, you must encode that requirement explicitly, using the `^` or `$` metacharacters. The above example, which is not so constrained, matches all these addresses:

```
bill@test.example
john@some.example
spoonbill@example.com
littlejohn@example.com
```

To match only the first two, you could use this:

```
if $sender_address matches "^(bill|john)@" then ...
```

Care must be taken if you need a backslash in a regular expression, because backslashes are interpreted as escape characters both by the string expansion code and by Exim’s normal processing of strings in quotes. For example, if you want to test the sender address for a domain ending in *.com* the regular expression is

```
\.com$
```

The backslash and dollar sign in that expression have to be escaped when used in a filter command, as otherwise they would be interpreted by the expansion code. Thus, what you actually write is

```
if $sender_address matches \\.com\$
```

An alternative way of handling this is to make use of the `\N` expansion flag for suppressing expansion:

```
if $sender_address matches \N\.com$\N
```

Everything between the two occurrences of `\N` is copied without change by the string expander (and in fact you do not need the final one, because it is at the end of the string). If the regular expression is given in quotes (mandatory only if it contains white space) you have to write either

```
if $sender_address matches "\\\\.com\\$"
```

or

```
if $sender_address matches "\\N\\.com$\\N"
```

If the regular expression contains bracketed sub-expressions, numeric variable substitutions such as *$1* can be used in the subsequent actions after a successful match. If the match fails, the values of the numeric variables remain unchanged. Previous values are not restored after *endif*. In other words, only one set of values is ever available. If the condition contains several sub-conditions connected by *and* or *or*, it is the strings extracted from the last successful match that are available in subsequent actions. Numeric variables from any one sub-condition are also available for use in subsequent sub-conditions, because string expansion of a condition occurs just before it is tested.

3.23 Numeric testing conditions

The following conditions are available for performing numerical tests:

```
      <number1> is above <number2>
      <number1> is not above <number2>
      <number1> is below <number2>
      <number1> is not below <number2>
e.g. $message_size is not above 10k
```

The <*number*> arguments must expand to strings of digits, optionally followed by one of the letters K or M (upper case or lower case) which cause multiplication by 1024 and 1024x1024 respectively.

3.24 Testing for significant deliveries

You can use the *delivered* condition to test whether or not any previously obeyed filter commands have set up a significant delivery. For example:

```
if not delivered then save mail/anomalous endif
```

“Delivered” is perhaps a poor choice of name for this condition, because the message has not actually been delivered; rather, a delivery has been set up for later processing.

3.25 Testing for error messages

The condition *error_message* is true if the incoming message is a bounce (mail delivery error) message. Putting the command

```
if error_message then finish endif
```

at the head of your filter file is a useful insurance against things going wrong in such a way that you cannot receive delivery error reports. **Note**: *error_message* is a condition, not an expansion variable, and therefore is not preceded by `$`.

3.26 Testing a list of addresses

There is a facility for looping through a list of addresses and applying a condition to each of them. It takes the form

```
foranyaddress <string> (<condition>)
```

where <*string*> is interpreted as a list of **RFC 2822 (https://datatracker.ietf.org/doc/html/rfc2822)** addresses, as in a typical header line, and <*condition*> is any valid filter condition or combination of conditions. The "group" syntax that is defined for certain header lines that contain addresses is supported.

The parentheses surrounding the condition are mandatory, to delimit it from possible further sub-conditions of the enclosing *if* command. Within the condition, the expansion variable *$thisaddress* is set to the non-comment portion of each of the addresses in the string in turn. For example, if the string is

```
B.Simpson <bart@sfld.example>, lisa@sfld.example (his sister)
```

then *$thisaddress* would take on the values `bart@sfld.example` and `lisa@sfld.example` in turn.

If there are no valid addresses in the list, the whole condition is false. If the internal condition is true for any one address, the overall condition is true and the loop ends. If the internal condition is false for all addresses in the list, the overall condition is false. This example tests for the presence of an eight-digit local part in any address in a *To:* header:

```
if foranyaddress $h_to: ( $thisaddress matches ^\\d{8}@ ) then ...
```

When the overall condition is true, the value of *$thisaddress* in the commands that follow *then* is the last value it took on inside the loop. At the end of the *if* command, the value of *$thisaddress* is reset to what it was before. It is best to avoid the use of multiple occurrences of *foranyaddress*, nested or otherwise, in a single *if* command, if the value of *$thisaddress* is to be used afterwards, because it isn't always clear what the value will be. Nested *if* commands should be used instead.

Header lines can be joined together if a check is to be applied to more than one of them. For example:

```
if foranyaddress $h_to:,$h_cc: ....
```

This scans through the addresses in both the *To:* and the *Cc:* headers.

3.27 Testing for personal mail

A common requirement is to distinguish between incoming personal mail and mail from a mailing list, or from a robot or other automatic process (for example, a bounce message). In particular, this test is normally required for "vacation messages".

The *personal* condition checks that the message is not a bounce message and that the current user's email address appears in the *To:* header. It also checks that the sender is not the current user or one of a number of common daemons, and that there are no header lines starting *List-* in the message. Finally, it checks the content of the *Precedence:* header line, if there is one.

You should always use the *personal* condition when generating automatic responses. This example shows the use of *personal* in a filter file that is sending out vacation messages:

```
if personal then
mail to $reply_address
subject "I am on holiday"
file $home/vacation/message
once $home/vacation/once
once_repeat 10d
endif
```

It is tempting, when writing commands like the above, to quote the original subject in the reply. For example:

```
subject "Re: $h_subject:"
```

There is a danger in doing this, however. It may allow a third party to subscribe you to an opt-in mailing list, provided that the list accepts bounce messages as subscription confirmations. (Messages sent from filters are always sent as bounce messages.) Well-managed lists require a non-bounce message to confirm a subscription, so the danger is relatively small.

If prefixes or suffixes are in use for local parts – something which depends on the configuration of Exim (see section 3.31 below) – the tests for the current user are done with the full address (including the prefix and suffix, if any) as well as with the prefix and suffix removed. If the system is configured to rewrite local parts of mail addresses, for example, to rewrite `dag46` as `Dirk.Gently`, the rewritten form of the address is also used in the tests.

3.28 Alias addresses for the personal condition

It is quite common for people who have mail accounts on a number of different systems to forward all their mail to one system, and in this case a check for personal mail should test all their various mail addresses. To allow for this, the *personal* condition keyword can be followed by

`alias <`*address*`>`

any number of times, for example:

```
if personal alias smith@else.where.example
            alias jones@other.place.example
then ...
```

The alias addresses are treated as alternatives to the current user's email address when testing the contents of header lines.

3.29 Details of the personal condition

The basic *personal* test is roughly equivalent to the following:

```
not error_message and
$message_headers does not contain "\nList-Id:" and
$message_headers does not contain "\nList-Help:" and
$message_headers does not contain "\nList-Subscribe:" and
$message_headers does not contain "\nList-Unsubscribe:" and
$message_headers does not contain "\nList-Post:" and
$message_headers does not contain "\nList-Owner:" and
$message_headers does not contain "\nList-Archive:" and
(
"${if def:h_auto-submitted:{present}{absent}}" is "absent" or
$header_auto-submitted: is "no"
) and
$header_precedence: does not contain "bulk" and
$header_precedence: does not contain "list" and
$header_precedence: does not contain "junk" and
foranyaddress $header_to:
( $thisaddress contains "$local_part$domain" ) and
not foranyaddress $header_from:
(
$thisaddress contains "$local_part@$domain" or
$thisaddress contains "server@" or
$thisaddress contains "daemon@" or
$thisaddress contains "root@" or
$thisaddress contains "listserv@" or
$thisaddress contains "majordomo@" or
$thisaddress contains "-request@" or
```

```
$thisaddress matches  "^owner-[^@]+@"
)
```

The variable *$local_part* contains the local part of the mail address of the user whose filter file is being run – it is normally your login id. The *$domain* variable contains the mail domain. As explained above, if aliases or rewriting are defined, or if prefixes or suffixes are in use, the tests for the current user are also done with alternative addresses.

3.30 Testing delivery status

There are two conditions that are intended mainly for use in system filter files, but which are available in users' filter files as well. The condition *first_delivery* is true if this is the first process that is attempting to deliver the message, and false otherwise. This indicator is not reset until the first delivery process successfully terminates; if there is a crash or a power failure (for example), the next delivery attempt is also a "first delivery".

In a user filter file *first_delivery* will be false if there was previously an error in the filter, or if a delivery for the user failed owing to, for example, a quota error, or if forwarding to a remote address was deferred for some reason.

The condition *manually_thawed* is true if the message was "frozen" for some reason, and was subsequently released by the system administrator. It is unlikely to be of use in users' filter files.

3.31 Multiple personal mailboxes

The system administrator can configure Exim so that users can set up variants on their email addresses and handle them separately. Consult your system administrator or local documentation to see if this facility is enabled on your system, and if so, what the details are.

The facility involves the use of a prefix or a suffix on an email address. For example, all mail addressed to *lg303-*<*something*> would be the property of user *lg303*, who could determine how it was to be handled, depending on the value of <*something*>.

There are two possible ways in which this can be set up. The first possibility is the use of multiple *.forward* files. In this case, mail to *lg303-foo*, for example, is handled by looking for a file called *.forward-foo* in *lg303*'s home directory. If such a file does not exist, delivery fails and the message is returned to its sender.

The alternative approach is to pass all messages through a single *.forward* file, which must be a filter file so that it can distinguish between the different cases by referencing the variables *$local_part_prefix* or *$local_part_suffix*, as in the final example in section 3.33 below.

It is possible to configure Exim to support both schemes at once. In this case, a specific *.forward-foo* file is first sought; if it is not found, the basic *.forward* file is used.

The *personal* test (see section 3.27) includes prefixes and suffixes in its checking.

3.32 Ignoring delivery errors

As was explained above, filtering just sets up addresses for delivery – no deliveries are actually done while a filter file is active. If any of the generated addresses subsequently suffers a delivery failure, an error message is generated in the normal way. However, if a filter command that sets up a delivery is preceded by the word "noerror", errors for that delivery, and any deliveries consequent on it (that is, from alias, forwarding, or filter files it invokes) are ignored.

3.33 Examples of Exim filter commands

Simple forwarding:

```
# Exim filter
deliver baggins@rivendell.middle-earth.example
```

Vacation handling using traditional means, assuming that the *.vacation.msg* and other files have been set up in your home directory:

```
# Exim filter
unseen pipe "/usr/ucb/vacation \"$local_part\""
```

Vacation handling inside Exim, having first created a file called *.vacation.msg* in your home directory:

```
# Exim filter
if personal then vacation endif
```

File some messages by subject:

```
# Exim filter
if $header_subject: contains "empire" or
$header_subject: contains "foundation"
then
save $home/mail/f+e
endif
```

Save all non-urgent messages by weekday:

```
# Exim filter
if $header_subject: does not contain "urgent" and
$tod_full matches "^(...),"
then
save $home/mail/$1
endif
```

Throw away all mail from one site, except from postmaster:

```
# Exim filter
if $reply_address contains "@spam.site.example" and
$reply_address does not contain "postmaster@"
then
seen finish
endif
```

Handle multiple personal mailboxes:

```
# Exim filter
if $local_part_suffix is "-foo"
then
save $home/mail/foo
elif $local_part_suffix is "-bar"
then
save $home/mail/bar
endif
```