University of Cambridge Computing Service

Specification of the Exim Mail Transfer Agent

by

Philip Hazel

University Computing Service
New Museums Site
Pembroke Street
Cambridge CB2 3QG
United Kingdom

phone: +44 1223 334600
fax: +44 1223 334679
email: ph10@cus.cam.ac.uk

Edition for Exim 3.00, May 1999

Contents

1. Introduction

If I have seen further it is by standing on the shoulders of giants. (Isaac Newton)

Exim is a mail transfer agent (MTA) for Unix systems connected to the Internet. Configuration files currently exist for the following operating systems: AIX, BSDI, DGUX, Digital UNIX, FreeBSD, HI-OSF (Hitachi), HP-UX, IRIX, Linux, MIPS RISCOS, NetBSD, OpenBSD, QNX, SCO, SCO SVR4.2 (aka UNIX-SV), SunOS4, Solaris (aka SunOS5), Ultrix, and Unixware. However, code is not available for determining system load averages under Ultrix.

The terms and conditions for the use and distribution of Exim are contained in the file **NOTICE**. Exim is distributed under the terms of the GNU General Public Licence, a copy of which may be found in the file **LICENCE**.

Exim owes a great deal to Smail 3 and its author, Ron Karr. Without the experience of running and working on the Smail 3 code, I could never have contemplated starting to write a new mailer. Many of the ideas and user interfaces are taken from Smail 3, though the actual code of Exim is entirely new.

I am indebted to my colleague Piete Brooks for suggesting and implementing the scheme for building Exim for multiple architectures and operating systems, for porting Exim to several different versions of Unix, and for numerous suggestions when I was first developing it. Many other people, both in Cambridge and around the world, have contributed to the development and the testing of Exim, and to porting it to various operating systems. I am grateful to them all.

This edition of the Exim specification applies to version 3.00 of Exim. Substantive changes from the 2.10 edition are marked by bars in the right margin, except in the Texinfo version of the documentation, because Texinfo doesn't support change bars. Minor corrections and rewordings are not so marked.

As the program is still developing, there may be features in later versions of the program that have not yet made it into this document, which is updated only when the most significant digit of the fractional part of the version number changes. However, all changes are noted briefly in the distributed file called **doc/ChangeLog**, and specifications of new features that are not yet in this manual are placed in **doc/NewStuff**.

1.1 Web site and Mailing list

There is a web site at **http://www.exim.org** by courtesy of Planet Online Ltd, who are situated in the UK. The site is mirrored in the USA at **http://www.us.exim.org** by courtesy of Shore.Net. Planet Online also run the following mailing lists:

exim-users@exim.org	general discussion list
exim-users-digest@exim.org	digest form of **exim-users**
exim-announce@exim.org	moderated, low volume announcements list

Messages that are sent to the announcements list are automatically copied to the main list, and thence to the digest list. You should therefore join only one list. Requests to be added to or deleted from the mailing lists should be sent to **exim-users-request@exim.org**, **exim-users-digest-request@exim.org**, or **exim-announce-request@exim.org**, respectively.

By courtesy of Martin Hamilton, there is an archive of the **exim-users** list in plain text form at **http://www.roads.lut.ac.uk/lists/exim-users/exim-users.archive** and in HTML via Hypermail at **http://www.roads.lut.ac.uk/lists/exim-users/**.

The list is also forwarded to **http://www.egroups.com/list/exim-users**, which is an archiving system with searching capabilities.

1.2 Availability

The current release of Exim is always to be found in

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/exim-*n.nn***.tar.gz**

where *n.nn* is the highest such version number in the directory. When there is only a small amount of change from one version to the next, a patch file may be provided, with a final component name of the form

exim-patch-*n.nn-m.mm***.gz**

For each released version, the log of changes is made separately available in the directory

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/ChangeLogs

so that it is possible to find out what has changed without having to download the entire distribution.

The main distribution contains ASCII versions of this specification and other documentation; other formats of the documents are available in separate files:

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/exim-html-*n.nn***.tar.gz**
ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/exim-pdf-*n.nn***.tar.gz**
ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/exim-postscript-*n.nn***.tar.gz**
ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/exim-texinfo-*n.nn***.tar.gz**

These tar files contain only the **/doc** directory, not the complete distribution.

An FAQ is available in two different formats from

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/FAQ.txt.gz
ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/FAQ.html.gz

The FAQ and other documentation is also available online at the web site.

There is a directory called

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/Contrib/

which contains miscellaneous files contributed to the Exim community by Exim users, and there is also a collection of contributed configuration examples in

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/config.samples.tar.gz

There are a number of sites that maintain mirrors of the Exim ftp directory. Those that I know about are listed in the file

ftp://ftp.cus.cam.ac.uk/pub/software/programs/exim/Mirrors

1.3 Limitations

- Exim is written in ANSI C. This should not be much of a limitation these days. However, to help with systems that lack a true ANSI C library, Exim avoids making any use of the value returned by the **sprintf()** function, which is one of the main incompatibilities. It has its own version of **strerror()** for use with SunOS4 and any other system that lacks this function, and a macro can be defined to turn **memmove()** into **bcopy()** if necessary.
- Exim uses file names that are longer than fourteen characters.
- Exim is intended for use as an Internet mailer, and therefore handles addresses in RFC 822 domain format only. It cannot handle ‘bang paths’, though simple two-component bang paths can be converted by a straightforward rewriting configuration.
- Exim insists that every address it handles has a domain attached. For incoming local messages, domainless addresses are automatically qualified with a configured domain value. Configuration options specify from which remote systems unqualified addresses are acceptable. These are then qualified on arrival.

- The only external transport currently implemented is an SMTP transport over a TCP/IP network (using sockets, including experimental support for IPv6). However, a pipe transport is available, and there are facilities for writing messages to files and pipes, optionally in *batched SMTP* format; these facilities can be used to send messages to some other transport mechanism such as UUCP, provided it can handle domain-style addresses. Batched SMTP input is also catered for.

1.4 Features

These are some of the main features of Exim:

- Exim follows the same general approach of decentralized control that Smail does. There is no central process doing overall management of mail delivery. However, unlike Smail, the independent delivery processes share data in the form of ‘hints’, which makes delivery more efficient in some cases.
- Exim has flexible retry algorithms, applicable to directing and routing addresses as well as to delivery.
- Exim contains header and envelope rewriting facilities.
- Unqualified addresses are accepted only from specified hosts or networks.
- Exim can perform multiple deliveries down the same SMTP channel after deliveries have been delayed.
- Exim can be configured to do local deliveries immediately but to leave remote (SMTP) deliveries until the message is picked up by a queue-runner process. This increases the likelihood of multiple messages being sent down a single SMTP connection.
- Remote deliveries of the same message to different hosts can optionally be done in parallel.
- Incoming SMTP messages start delivery as soon as they are received, without waiting for the SMTP call to close.
- Perl-compatible regular expressions are available in a number of configuration parameters.
- It is also possible to build Exim with an embedded Perl interpreter, allowing arbitrary Perl code to be run in certain circumstances.
- Domain lists can include file lookups, making it possible to support very large numbers of local domains.
- Exim supports optional checking of incoming return path (sender) and receiver addresses as they are received by SMTP.
- SMTP calls from specific machines, optionally from specific idents, can be locked out, and incoming SMTP messages from specific senders can also be locked out. Exim also supports the use of the Realtime Blocking List (RBL).
- Hosts that are permitted to relay mail through a machine to another external domain can be controlled by IP number or IP network number. Relay control by recipient domain and sender address is also available.
- Messages on the queue can be ‘frozen’ and ‘thawed’ by the administrator.
- Exim can handle a number of independent local domains on the same machine; each domain can have its own alias files, etc. This facility is sometimes known as ‘virtual domains’.
- Exim stats a user’s home directory before looking for a **.forward** file, in order to detect the case of a missing NFS mount. Delivery is delayed if the directory is unavailable.
- Exim contains an optional built-in mail filtering facility. This can be configured to allow users to provide personal filter files, and it is also possible for a system-wide filter file to be applied to every message.

- Periodic warnings are automatically sent to messages' senders when delivery is delayed – the time between warnings is configurable. The warnings can be made conditional on the contents of the message.
- A queue run can be manually started to deliver just a particular portion of the queue, or those messages with a recipient whose address contains a given string. There is support for the ETRN command in SMTP to interface to this.
- Exim can be configured to run as root all the time, except when performing local deliveries, which it always does in a separate process under an appropriate uid and gid. Alternatively, it can be configured to run as root only when needed; in particular, it need not run as root when receiving incoming messages or when sending out messages over SMTP. See chapter 49 for a discussion of security issues.
- I have tried to make the wording of delivery failure messages clearer and simpler, for the benefit of those less-experienced people who are now using email. Alternative wording for these messages can be provided in a separate file.
- The Exim Monitor is an optional extra; it displays information about Exim's processing in an X window, and an administrator can perform a number of control actions from the window interface. However, all such actions are also available from the command line interface.

1.5 Support for IPv6

The current IP protocol is more strictly called IPv4. IPv6 is the next generation of IP protocol; it is currently in an experimental state. A number of vendors have released IPv6 versions of their systems and libraries.

If Exim is built with HAVE_IPV6 set, it uses the IPv6 API for TCP/IP input and output. IP addresses can be given in IPv6 as well as IPv4 notation; incoming IPv4 calls use the embedded IPv6 address notation. Currently, Exim cannot discover for itself the addresses of the host's IPv6 interfaces (at least, not on Solaris 2 – I don't know the position on other systems) so the **local_interfaces** option must always be used to list them explicitly, in order to prevent mail looping.

See the file **README.IPV6** for general information about the current state of IPv6 support in Exim.

1.6 Calling interface

Like many MTAs, Exim has adopted the Sendmail interface so that it can be a straight replacement for **/usr/lib/sendmail**. All the relevant Sendmail options are implemented, with two reservations. There are also some additional options that are compatible with Smail 3, and some further options that are new to Exim.

The **-t** option, for taking a list of recipients from a message's headers, is documented (for Sendmail) as suppressing delivery to any addresses on the command line. However, it appears that this is not the case in practice. For this reason, Exim has an option called **extract_addresses_remove_arguments** which controls its behaviour in this regard.

Sendmail uses the **-bi** option as a request to rebuild the alias file. As Exim does not have the concept of a single alias file, it cannot mimic this behaviour. It can be configured to run a particular script when this option is received; otherwise the option is ignored.

The runtime configuration is held in a single text file which is divided into a number of sections. The entries in this file consist of keywords and values, in the style of Smail 3 configuration files. A default configuration file which is suitable for simple installations is provided in the distribution.

Control of messages on the queue can be done via certain privileged command line options. There is also an optional monitor program called **eximon**, which displays current information in an X window, and contains a menu interface to Exim's command line administration options.

1.7 Terminology

The term *local part*, which is taken from RFC 822, is used to refer to that part of an email address that precedes the @ sign. The part that follows the @ sign is called the *domain* or *mail domain*.

The word *domain* is sometimes used to mean all but the first component of a machine's name. It is *not* used in that sense here, where it normally refers to the part of an email address following the @ sign.

Local domains are mail domains for which the current host is responsible; in other words, it has special knowledge of what to do with messages sent to such domains, and normally that means using the local part of the address either to deliver the message on the local host or to transform the address using an alias file or something similar. All other domains are *remote domains*, whose appearance normally causes the message to be transmitted to some other host.

The distinction between local and remote domains is not always entirely clear-cut, since a host can have special knowledge about routing for remote domains, and messages for local domains may under some circumstances be passed to other hosts.

The terms *local delivery* and *remote delivery* are used to distinguish delivery to a file or a pipe on the local machine from delivery by SMTP to some remote machine. The type of delivery does not necessarily correspond to the type of address. Mail for a local domain may get passed on to some other host, while mail for a remote domain might get delivered locally to a file or pipe for onward transmission by some other means. However, these are special cases.

The term *mailmaster* is used to refer to the person in charge of maintaining the mail software on a given computer. Commonly this will be the same person who fulfils the postmaster role, but this may not always be the case.

The term *queue* is used to refer to the set of messages awaiting delivery, because this term is in widespread use in the context of MTAs. However, in Exim's case the reality is more like a pool than a queue, because there is normally no ordering of waiting messages.

The term *queue-runner* is used to describe a process that scans the queue and attempts to deliver those messages whose retry times have come. This term is used by other MTAs, and also relates to the command **runq**, but in Exim the waiting messages are normally processed in an unpredictable order.

2. Incorporated code

A number of pieces of external code are included in the Exim distribution.

- Regular expressions are supported in the main Exim program and in the Exim monitor using the freely-distributable PCRE library, copyright © 1999 University of Cambridge. The source is distributed in the directory **src/pcre**.
- RFC 1413 callbacks are supported in the main Exim program using the **libident** library made freely available by Peter Eriksson at **ftp.lysator.liu.se**. Some modifications have been made in order to support IPv6. The source is distributed in the directory called **src/libident**.
- Support for the cdb (Constant DataBase) lookup method is provided by code contributed by Nigel Metheringham of Planet Online Ltd. which contains the following statements:

  The implementation is completely contained within the code of Exim. It does not link against an external cdb library.
- The Exim Monitor program, which is an X-Window application, includes modified versions of the Athena StripChart and TextPop widgets. This code is copyright by DEC and MIT, and their permission notice appears below, in accordance with the conditions expressed therein.

[6] *incorporated code (2)*

3. How Exim delivers mail

3.1 Philosophy

Exim is designed to work efficiently on systems that are permanently connected to the Internet and are handling a general mix of mail. In such circumstances, most messages can be delivered immediately. Consequently, Exim does not maintain independent queues of messages for specific domains or hosts, though it does try to send several messages in a single SMTP connection after a host has been down, and it also maintains per-host retry information.

3.2 Message reception

When Exim receives a message, it writes two files in its spool directory. The first contains the *envelope* information, the current status of the message, and the headers, while the second contains the body of the message.

The envelope information consists of the address of the message's sender and the address(es) of the recipient(s). This information is entirely separate from any addresses contained in the headers. The status of the message includes a list of recipients who have already received the message. The format of the first spool file is described in chapter 50.

Any header rewriting that is specified in the configuration (see chapter 32) is done once and for all at the time the message is received. It is also possible to specify the addition or removal of certain headers at the time the message is delivered (see chapters 14 and 19).

Every message handled by Exim is given a *message id* which is sixteen characters long. It is divided into three parts, separated by hyphens. Each part is a sequence of letters and digits, representing a number in base 62:

- The first six characters are the time the message was received, as a number in seconds – the normal Unix way of representing a time of day. If the clock goes backwards (due to resetting) in a process that is receiving more than one message, the later time is retained.
- After the first hyphen, the next six characters are the id of the process that received the message.
- The final two characters, after the second hyphen, are used to ensure uniqueness of the id. There are two different formats:

  (a) If the **host_number** option is not set, uniqueness is required only within the local host. This portion of the id is '00' except when a process receives more than one message in a single second, when the number is incremented for each additional message.

  (b) If the **host_number** option is set, uniqueness among a set of hosts is required. This portion of the id is set to the base 62 encoding of

  <*sequence number*> `* 256 +` <*host number*>

  where <*sequence number*> is the count of messages received by the current process within the current second. As the maximum value of the host number is 255, this allows for a maximum value of 14 for the sequence number. If this limit is reached, a delay of one second is imposed before reading the next message, in order to allow the clock to tick and the sequence number to get reset.

The names of the two spool files consist of the message id, followed by `-H` for the file containing the envelope and headers, and `-D` for the data file.

By default all these spool files are held in a single directory called **input** inside the general Exim spool directory. Some operating systems do not perform very well if the number of files in a directory gets very large; to improve performance in such cases there is an option that causes Exim to split up the input files into 62 sub-directories whose names are single letters or digits.

Exim can be configured not to start a delivery process automatically when a message is received; this can be unconditional, or depend on the number of incoming SMTP connections or the system load (where available). In these situations, new messages wait on the queue until a queue-runner process picks them up, but normally delivery is started as soon as a message is received.

3.3 Life of a message

A message remains in the spool directory until it is completely delivered to its recipients or to an error address, or until it is deleted by an administrator or by the user who originally created it. In cases when delivery cannot proceed – for example, when a message can neither be delivered to its recipients nor returned to its sender, the message is marked 'frozen' on the spool, and no more deliveries are attempted.

An administrator can 'thaw' such messages when the problem has been corrected, and can also freeze individual messages by hand if necessary. In addition, an administrator can force a delivery error, causing an error message to be sent.

There is also an **auto_thaw** option, which can be used to cause Exim to retry frozen messages after a certain time. When this is set, no message will remain on the queue for ever, because the delivery timeout will eventually be reached. Delivery failure reports that reach this timeout are discarded.

As delivery proceeds, Exim writes timestamped information about each address to a per-message log file; this includes any delivery error messages. This log is solely for the benefit of the administrator, and is normally deleted with the spool files when processing of a message is complete. However, Exim can be configured to retain it (a dangerous option, as the files can accumulate rapidly on a busy system). Exim also writes delivery messages to its main log file, whose contents are described in chapter 45.

All the information Exim itself needs to set up a delivery is kept in the first spool file with the headers. When a successful delivery occurs, the address is immediately written at the end of a journal file, whose name is the message id followed by `-J`. At the end of a delivery run, if there are some addresses left to be tried again later, the first spool file is updated to indicate which these are, and the journal file is then deleted. Updating the spool file is done by writing a new file and renaming it, to minimize the possibility of data loss.

Should the system or the program crash after a successful delivery but before the spool file has been updated, the journal is left lying around. The next time Exim attempts to deliver the message, it reads the journal file and updates the spool file before proceeding. This minimizes the chances of double deliveries caused by crashes.

3.4 Drivers

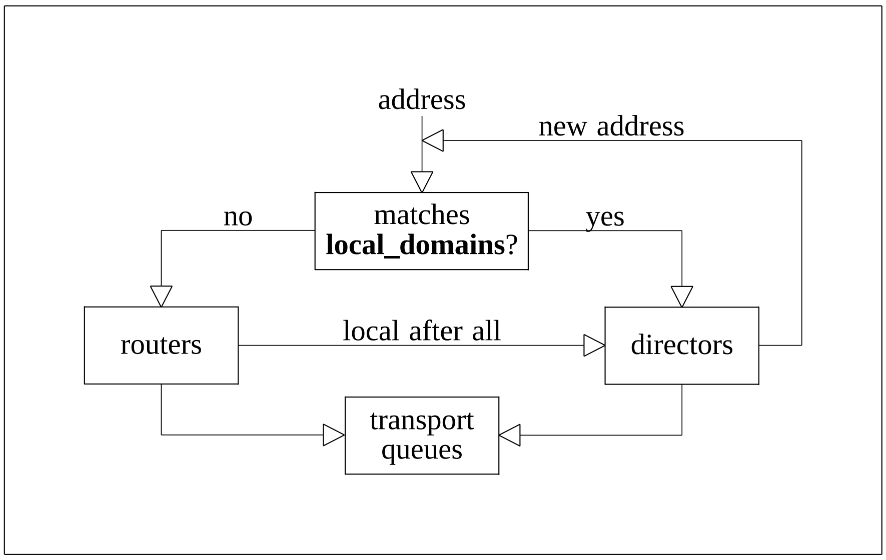
The main delivery processing elements of Exim are called *directors*, *routers*, and *transports*, and collectively these are known as *drivers*. Code for a number of them is provided, compile-time options specify which ones are included in the binary, and runtime options specify which ones are actually used.

A *transport* is a driver that transmits a copy of the message from Exim's spool to some destination. There are two kinds of transport: for a *local* transport, the destination is a file or a pipe on the local host, while for a *remote* transport the destination is some other host. A message is passed to a specific transport as a result of successful directing or routing. If a message has several recipients, it may be passed to a number of different transports.

A *director* is a driver that operates on a local address, either determining how its delivery should happen, or converting the address into one or more new addresses (for example, via an alias file). A local address is one whose domain matches an entry in the list given in the **local_domains** option, or has been determined to be local by a router – see below. The fact that an address is local does not imply that the message has to be delivered locally; it can be directed either to a local or to a remote transport.

A *router* is a driver that operates on an apparently remote address, that is an address whose domain does not match anything in the list given in **local_domains**. When a router succeeds it can route an address either to a local or to a remote transport, or it can change the domain, and pass the address on to subsequent routers.

In exceptional cases, a router may determine that an address is local after all, and cause it to be passed to the directors. This happens automatically if a host lookup expands an abbreviated domain into one that is local. It can also be made to happen (optionally) if an MX record or other routing information points to the local host, though by default this situation is treated as a configuration error. This is the only case in which the directors are used to process an address that may not match anything in **local_domains**. The diagram below illustrates the relationship between the three kinds of driver.



Driver interactions

As new features have been added to Exim, the distinction between routers and directors has become less clear-cut than it once was. However, since a typical configuration has a number of directors and routers, checking the domain against **local_domains** once at the start does use fewer resources than checking it for each of them.

3.5 Delivery in detail

When a message is to be delivered, the sequence of events is roughly as follows:

- If a system-wide filter file is specified, the message is passed to it. The filter may add recipients to the message, replace the recipients, discard the message, cause a new message to be generated, or cause the message delivery to fail. This facility is intended as a weapon against mail bombs and unsolicited mail. The format of the filter file is the same as for user filter files, described in the separate document entitled *Exim's User interface to mail filtering*. Some additional features are available in system filters – see chapter 41 for details. Note that a message is passed to the system filter only once per delivery attempt, however many recipients it has. However, if there are several delivery attempts because one or more addresses could not be immediately delivered, the system filter is run each time. The filter condition **first_delivery** can be used to detect this.
- Each recipient address is parsed and a check is made to see if it is local, by comparing the domain with the list in the **local_domains** option. This can contain wildcards and file lookups.

- If an address is local, it is passed to each configured director in turn until one is able to handle it. If none can, the address is failed. Directors can be targeted at particular local domains, so several local domains can be processed entirely independently of each other.
- A director that accepts an address may set up a local or a remote transport for it. The transport is not run at this time; the address is placed on a queue for the particular transport, to be run later. Alternatively, the director may generate one or more new addresses (typically from alias, forward, or filter files). New addresses are fed back into this process from the top, but in order to avoid loops, a director ignores any address which has an identically-named ancestor that was processed by itself.
- If an address is not local, it is passed to each configured router in turn until one is able to handle it. If none can, the address is failed.
- A router that accepts an address may set up a transport for it, or may pass an altered address to subsequent routers, or it may discover that the address is a local address after all. This typically happens when a partial domain name is used and (for example) the DNS lookup is configured to try to extend such names. In this case, the address is passed to the directors. Exim can also be configured to do this for any domain whose lowest MX record or other routing information points to the local host.
- Routers normally set up remote transports for messages that are to be delivered to other machines. However, a router can pass a message to a local transport, and by this means such messages can be routed to other transport mechanisms via pipes or files.
- When all the directing and routing is done, addresses that have been successfully handled are passed to their assigned transports. When local transports are doing real local deliveries, they handle only one address at a time, but if a local transport is being used as a pseudo-remote transport (for example, to collect batched SMTP messages for transmission by some other means) multiple addresses can be handled. Remote transports can always handle more than one address at once, but can be configured not to do so, or to restrict multiple addresses to the same domain.
- Each local delivery runs in a separate process under a non-privileged uid, and they are run in sequence. Exim can be configured so that remote deliveries run under a uid that is private to Exim, instead of running as root. By default the remote deliveries run one at a time in the main Exim process, but a configuration option is available to allow multiple remote deliveries for a single message to be run simultaneously, each in its own sub-process.
- When it is doing a queue run, Exim checks its retry database to see if there has been a previous temporary delivery failure for the address before running any local transport. If it finds one, it does not attempt a new delivery until the retry time for the address is reached. However, this happens only for delivery attempts that are part of a queue run, so local deliveries are always attempted when delivery immediately follows message reception.
- Remote transports do their own retry handling, since an address may be deliverable to one of a number of hosts, each of which may have a different retry time. If there have been previous failures and no host has reached its retry time, no delivery is attempted, whether in a queue run or not. See chapter 31 for details of retry strategies.
- If there were any errors, a message is returned to an appropriate address (the sender in the common case), with details of the error for each failing address. Exim can be configured to send copies of error messages to other addresses.
- If one or more addresses suffered a temporary failure, the message is left on the queue, to be tried again later. Otherwise the spool files and message log are deleted, though the message log can optionally be preserved if required.

Delivery is said to be *deferred* when the message remains on the queue for a subsequent delivery attempt after a temporary failure. Such messages get processed again by queue-runner processes that are periodically started, either by an Exim daemon or via **cron**.

Temporary failures may be detected during routing and directing as well as during the transport stage. Exim uses a set of configured rules to determine when next to retry the failing address (see chapter 31). These rules also specify when Exim should give up trying to deliver to the address, at which point it generates a failure report.

When a delivery is not part of a queue run (typically an immediate delivery on receipt of a message), the directors are always run for local addresses, and local deliveries are always attempted, even if retry times are set for them. This makes for better behaviour if one particular message is causing problems (for example, causing quota overflow, or provoking an error in a filter file). If such a delivery suffers a temporary failure, the retry data gets updated as usual, for use by the next queue-runner process.

When a message cannot be delivered to some or all of its intended recipients, a delivery failure report is generated. All the addresses that failed in a given delivery attempt are listed in a single failure report. If a message has many recipients, it is possible for some addresses to fail in one delivery attempt and others to fail subsequently, giving rise to more than one failure report for a single message. The wording of delivery failure reports can be customized by the administrator. See chapter 33 for details.

Delivery failure messages contain an **X-Failed-Recipients:** header, listing all failed addresses, for the benefit of programs that try to analyse such messages automatically.

A failure report is normally sent to the sender of the original message, as obtained from the message's envelope. For incoming SMTP messages, this is the address given in the MAIL command. However, when an address is expanded via a forward or alias file, an alternative address can be specified for delivery failures of the generated addresses. For a mailing list expansion (see chapter 36) it is common to direct failure reports to the manager of the list.

If a failure report (either locally generated or received from a remote host) itself suffers a delivery failure, the message is left on the queue, but is 'frozen', awaiting the attention of an administrator. There are options which can be used to make Exim discard such failure reports, or to keep them for only a short time.

3.6 Temporary delivery failures

There are many reasons why a message may not be immediately deliverable to a particular address. Failure to connect to a remote machine (because it, or the connection to it, is down) is one of the most common. Local deliveries may also be delayed if NFS files are unavailable, or if a mailbox is on a file system where the user is over quota. Exim can be configured to impose its own quotas on local mailboxes; where system quotas are set they will also apply.

A machine that is connected to the Internet can normally deliver most mail straight away (the usual figure for Cambridge is 98%). In its default configuration, Exim starts a delivery process whenever it receives a message, and usually this completes the entire delivery. This is a lightweight approach, avoiding the need for any centralized queue managing software. There are those who argue that a central message manager would be able to batch up messages for the same host and send them in a single SMTP call. I do not myself believe this would occur much in general, unless messages were significantly delayed in order to create a batch.

However, if a host is unreachable for a period of time, a number of messages may be waiting for it by the time it recovers, and sending them in a single SMTP connection is clearly beneficial. Whenever a delivery to a remote host is deferred, Exim makes a note in its hints database, and whenever a successful SMTP delivery has happened, it looks to see if any other messages are waiting for the same host. If any are found, they are sent over the same SMTP connection, subject to a configuration limit as to the maximum number in any one connection.

4. Building and installing Exim

4.1 Unpacking

Exim is distributed as a gzipped tar file which, when upacked, creates a directory with the name of the current release (for example, **exim-3.00**) into which the following files are placed:

LICENCE	the GNU General Public Licence
Makefile	top-level make file
NOTICE	conditions for the use of Exim
README	list of files, directories and simple build instructions

Other files whose names begin with **README** may also be present. The following subdirectories are created:

OS	OS-specific files
doc	documentation files
exim_monitor	source files for the Exim monitor
scripts	scripts used in the build process
src	remaining source files
util	independent utilities

Some utilities are contained in the **src** directory, and are built with the Exim binary; those distributed in the **util** directory are things like the log file analyser, which do not depend on any compile-time configuration.

4.2 Multiple machine architectures and operating systems

The building process for Exim is arranged to make it easy to build binaries for a number of different architectures and operating systems from the same set of source files. Compilation does not take place in the **src** directory. Instead, a *build directory* is created for each architecture and operating system. Symbolic links to the sources are installed in this directory, which is where the actual building takes place.

In most cases, Exim can discover the machine architecture and operating system for itself, but the defaults can be overridden if necessary.

4.3 DBM libraries

Licensed versions of Unix normally contain a library of DBM functions operating via the 'ndbm' interface, and this is what Exim expects by default. Free versions of Unix seem to vary in what they contain as standard. In particular, some versions of Linux have no default DBM library, and different distributors have chosen to bundle different libraries with their packaged versions. However, the more recent releases seem to have standardised on the Berkeley DB library.

Different DBM libraries have different conventions for naming the files they use. When a program opens a file called **dbmfile**, there are four possibilities:

(1) A traditional ndbm implementation, such as that supplied as part of Solaris 2, operates on two files called **dbmfile.dir** and **dbmfile.pag**.

(2) The GNU library, **gdbm**, operates on a single file, but makes two different hard links to it with names **dbmfile.dir** and **dbmfile.pag**.

(3) The Berkeley DB package, if called via its ndbm compatibility interface, operates on a single file called **dbmfile.db**, but otherwise looks to the programmer exactly the same as the traditional ndbm implementation.

(4) If the Berkeley package is used in its native mode, it operates on a single file called **dbmfile**; the programmer's interface is somewhat different to the traditional ndbm interface.

Exim and its utilities can be compiled to use any of these interfaces. By default it assumes an interface of type (1), though some operating system configuration files default to assuming (4). In order to use the Berkeley DB package in native mode, it is necessary to set USE_DB in an appropriate configuration file, and it may also be necessary to set DBMLIB, for example,

```
DBMLIB = -ldb
```

By avoiding the translation from one interface to another, some resources may be saved.

To complicate things further, there are now two very different versions of the Berkeley DB package. Version 1.85 has been stable for quite some time, but the latest versions are numbered 2.x. Release 2 is very different internally and externally from the 1.85 release. Both versions of Berkeley DB can be obtained from

```
http://www.sleepycat.com/
```

but maintenance of version 1.85 is being phased out, and it may not compile on some systems. There is further discussion about the various DBM libraries in the file **doc/dbm.discuss.txt**.

4.4 Pre-building configuration

Before building Exim, a local configuration file that specifies options independent of any operating system has to be created with the name **Local/Makefile**. A template for this file is supplied as the file **src/EDITME**, and it contains full descriptions of all the option settings therein.

Default values are supplied for all of them except for those that specify the locations of the runtime configuration file and the directory for holding Exim binaries. These must be given, as Exim will not build without them. There are a few parameters that can be specified either at build time or at run time to enable the same binary to be used on a number of different machines. However, if the locations of Exim's spool directory and log file directory (if not within the spool directory) are fixed, it is recommended that you specify them in **Local/Makefile** instead of at run time so that errors detected early in Exim's execution (such as a malformed configuration file) can be logged.

If you are going to build the Exim monitor, a similar configuration process is required. The file **exim_monitor/EDITME** must be edited appropriately for your installation and saved under the name **Local/eximon.conf**. If you are happy with the default settings described in **exim_monitor/EDITME**, then **Local/eximon.conf** can be empty, but it must exist.

This is all the configuration that is needed in straightforward cases for known operating systems. However, the building process is set up so that it is easy to override options that are set by default or by operating-system-specific configuration files, for example to change the name of the C compiler, which defaults to **gcc**. See section 4.8 below for details of how to do this.

4.5 Use of tcpwrappers

Exim can be linked with the **tcpwrappers** library in order to check incoming SMTP calls using the **tcpwrappers** control files. This may be a convenient alternative to Exim's own checking facilities for installations that are already making use of **tcpwrappers** for other purposes. To do this, you should set USE_TCP_WRAPPERS in **Local/Makefile**, arrange for the file **tcpd.h** to be available at compile time, and also ensure that the library **libwrap.a** is available at link time, typically by including '-lwrap' in EXTRALIBS. For example, if **tcpwrappers** is installed in **/usr/local**, you might have

```
USE_TCP_WRAPPERS=yes
CFLAGS=-O -I/usr/local/include
EXTRALIBS=-L/usr/local/lib -lwrap
```

in **Local/Makefile**. The name to use in the **tcpwrappers** control files is 'exim'. For example, the line

```
exim : LOCAL  192.168.0.  .friendly.domain
```

in your **/etc/hosts.allow** file allows connections from the local host, from the subnet 192.168.0.0/24, and from all hosts in ***.friendly.domain**. All other connections are denied. Consult the **tcpwrappers** documentation for further details.

4.6 Including support for IPv6

Exim contains experimental code for use on systems that have IPv6 support. The file **README.IPV6** contains information on the current status of IPv6 in Exim. Setting HAVE_IPV6=YES in **Local/Makefile** causes the IPv6 code to be included; it may also be necessary to set IPV6_INCLUDE and IPV6_LIBS.

4.7 The building process

Once **Local/Makefile** (and **Local/eximon.conf**, if required) have been created, run *make* at the top level. It determines the architecture and operating system types, and creates a build directory if one does not exist. For example, on a Sun system running Solaris 2.5.1, the directory **build-SunOS5-5.5.1-sparc** is created. Symbolic links to relevant source files are installed in the build directory.

If this is the first time *make* has been run, it calls a script which builds a make file inside the build directory, using the configuration files from the **Local** directory. The new make file is then passed to another instance of *make* which does the real work, building a number of utility scripts, and then compiling and linking the binaries for the Exim monitor (if configured), a number of utilities, and finally Exim itself. The command *make makefile* can be used to force a rebuild of the make file in the build directory, should this ever be necessary.

If you have problems building Exim, check for any comments there may be in the **README** file concerning your operating system, and also take a look at the FAQ, where some common problems are covered.

4.8 Overriding build-time options for Exim

The main make file that is created at the beginning of the building process consists of the concatenation of a number of files which set configuration values, followed by a fixed set of *make* instructions. If a value is set more than once, the last setting overrides any previous ones. This provides a convenient way of overriding defaults. The files that are concatenated are, in order:

OS/Makefile-Default
OS/Makefile-<*ostype*>
Local/Makefile
Local/Makefile-<*ostype*>
Local/Makefile-<*archtype*>
Local/Makefile-<*ostype*>-<*archtype*>
OS/Makefile-Base

where <*ostype*> is the operating system type and <*archtype*> is the architecture type. **Local/Makefile** is required to exist, and the building process fails if it is absent. The other three **Local** files are optional, and are often not needed.

The values used for <*ostype*> and <*archtype*> are obtained from scripts called **scripts/os-type** and **scripts/arch-type** respectively. If either of the environment variables OSTYPE or ARCHTYPE is set, their values are used, thereby providing a means of forcing particular settings. Otherwise, the scripts try various *ad hoc* methods of determining these values. You can run these scripts directly from the shell in order to find out what values are being used on your system.

OS/Makefile-Default contains comments about the variables that are set therein. Some (but not all) are mentioned below. If there is something that needs changing, review the contents of this file and the contents of the make file for your operating system (**OS/Makefile-<*ostype*>**) to see what the default values are.

If you need to change any of the values that are set in **OS/Makefile-Default** or in **OS/Makefile-<*ostype*>**, or to add any new definitions, do so by putting the new values in an appropriate **Local** file. For example, to specify that the C compiler is called **cc** rather than **gcc** when compiling in the OSF1 operating system, and that it is to be to be called with the flag **-std1**, create a file called **Local/Makefile-OSF1** containing the lines

```
CC=cc
CFLAGS=-std1
```

This makes it easy to transfer your configuration settings to new versions of Exim simply by copying the contents of the **Local** directory.

Exim contains support for doing LDAP, NIS, and NIS+ lookups, but not all systems have these components installed, so the default is not to include the relevant code in the binary. All the different kinds of file and database lookup that Exim supports are implemented as separate code modules which are included only if the relevant compile-time options are set. In the case of LDAP, NIS, and NIS+, the settings for **Local/Makefile** are:

```
LOOKUP_LDAP=yes
LOOKUP_NIS=yes
LOOKUP_NISPLUS=yes
```

In all cases the relevant include files and interface libraries need to be installed before compiling Exim. When a lookup type is not included in the binary, attempts to configure Exim to use it cause configuration errors.

Another optional lookup type is cdb, which is included in the binary only if

```
LOOKUP_CDB=yes
```

is set. In this case, the code is entirely contained within Exim, and no external include files or libraries are required.

Exim can be linked with an embedded Perl interpreter, allowing Perl subroutines to be called during string expansion. To enable this facility,

```
EXIM_PERL=perl.o
```

must be defined in **Local/Makefile**. Details of this facility are given in chapter 10.

The location of the X11 libraries is something that varies a lot between operating systems, and of course there are different versions of X11 to cope with. The following three variables are set in **OS/Makefile-Default**:

```
X11=/usr/X11R5
XINCLUDE=-I$(X11)/include
XLFLAGS=-L$(X11)/lib
```

These are overridden in some of the operating-system configuration files. For example, in **OS/Makefile-SunOS5** there is

```
X11=/usr/openwin
XINCLUDE=-I$(X11)/include
XLFLAGS=-L$(X11)/lib -R$(X11)/lib
```

If you need to override the default setting for your operating system, place a definition of all three of these variables into your **Local/Makefile-**<*ostype*> file.

If you need to add any extra libraries to the link steps, these can be put in a variable called EXTRALIBS, which appears in all the link commands, but by default is not defined. There is also DBMLIB, which appears in the link commands for binaries that use DBM functions (see also section 4.3). Finally, there is EXTRALIBS_EXIMON, which appears only in the link step for the Exim monitor binary, and which can be used, for example, to include additional X11 libraries.

Another variable which is not normally defined is STDERR_FILE. This defines a file to which debugging output is written if the **-df** flag is set, and is of use when running Exim under **inetd**.

Yet another variable which should not normally be needed is ERRNO_QUOTA. Exim needs to know which error the operating system gives when writing to a file fails because the user is over quota. POSIX specifies an error called EDQUOT and this is present in the latest versions of all the systems Exim has been ported to at the time of writing. However, it is not present in earlier versions of SunOS5, which use ENOSPC instead. The code of Exim defaults to using EDQUOT if it is defined, and ENOSPC otherwise. You should set ERRNO_QUOTA only if your system uses some completely different error code.

The make file copes with rebuilding Exim correctly if any of the configuration files are edited. However, if an optional configuration file is deleted, it is necessary to touch the associated non-optional file (that is, **Local/Makefile** or **Local/eximon.conf**) before rebuilding.

4.9 OS-specific header files

The **OS** directory contains a number of files with names of the form **os.h-**<*ostype*>. These are system-specific C header files that should not normally need to be changed. There is a list of macro settings that are recognized in the file **OS/os.configuring**, which should be consulted if you are porting Exim to a new operating system.

4.10 Overriding build-time options for the monitor

A similar process is used for overriding things when building the Exim monitor, where the files that are involved are

OS/eximon.conf-Default
OS/eximon.conf-<*ostype*>
Local/eximon.conf
Local/eximon.conf-<*ostype*>
Local/eximon.conf-<*archtype*>
Local/eximon.conf-<*ostype*>**-**<*archtype*>

As with Exim itself, the final three files need not exist, and in this case the **OS/eximon.conf-**<*ostype*> file is also optional. The default values in **OS/eximon.conf-Default** can be overridden dynamically by setting environment variables of the same name, preceded by EXIMON_. For example, setting EXIMON_LOG_DEPTH in the environment overrides the value of LOG_DEPTH at run time.

4.11 Installing commands and scripts

The script **scripts/exim_install** copies binaries and utility scripts into the directory whose name is specified by the BIN_DIRECTORY setting in **Local/Makefile**. Files are copied only if they are newer than any versions already in the binary directory, and old versions are renamed by adding the suffix **.O** to their names.

The command **make install** runs the **exim_install** script with no arguments. It can be run independently with arguments specifying which files are to be copied, from within a build directory. For example,

```
(cd build-SunOS5-sparc; ../scripts/exim_install exim)
```

copies just the main binary file. The main **exim** binary is required to be owned by root and setuid, for normal configurations. In some special cases (for example, if a host is doing no local deliveries) is is possible to run Exim in other ways. If the binary is run by a root process, the effect is the same as if it were setuid root. The install script tries to set root as the owner of the main binary, and to make it setuid. It should therefore normally be run as root. If you want to see what the script will do before running it for real, use the **-n** option (for which root is not needed):

```
(cd build-SunOS5-5.5.1-sparc; ../scripts/exim_install -n)
```

Exim's runtime configuration file is named by the CONFIGURE_FILE setting in **Local/Makefile**. If this file does not exist, then the default configuration file **src/configure.default** is copied there by the installation script. If a runtime configuration file already exists, it is left alone. The default configuration uses the local host's name as the only local domain, and is set up to do local deliveries into the shared directory **/var/mail**, running as the local user. Aliases in **/etc/aliases** and **.forward** files in users' home directories are supported, but no NIS or NIS+ support is configured. Remote domains are routed using the DNS, with delivery over SMTP.

4.12 Installing info documentation

Not all systems use the GNU **info** system for documentation, and for this reason, the Texinfo source of Exim's documentation is not included in the main distribution. Instead it is available separately from the ftp site (see section 1.2).

If you have defined INFO_DIRECTORY in **Local/Makefile** and the Texinfo source of the documentation is found in the source tree, then running **make install** automatically builds the info files and installs them.

4.13 Setting up the spool directory

When it starts up, Exim tries to create its spool directory if it does not exist. If a specific Exim uid and gid are specified, these are used for the owner and group of the spool directory. Sub-directories are automatically created in the spool directory as necessary.

4.14 Testing

Having installed Exim, you can check that the runtime configuration file is syntactically valid by running the command

```
exim -bV
```

If there are any errors in the configuration file, Exim will output error messages. Otherwise it just outputs the version number and build date. Some simple routing tests can be done by using the address testing option. For example,

```
exim -v -bt <local username>
```

should verify that it recognizes a local mailbox, and

```
exim -v -bt <remote address>
```

a remote one. Then try getting it to deliver mail, both locally and remotely. This can be done by passing messages directly to Exim, without going through a user agent. For example:

```
exim postmaster@your.domain
From: user@your.domain
To: postmaster@your.domain
Subject: Testing Exim

This is a test message.
^D
```

If you encounter problems, look at Exim's log files (**mainlog** and **paniclog**) to see if there is any relevant information there. Another source of information is running Exim with debugging turned on, by specifying the **-d** option. The larger the number after **-d** (up to 9), the more information is output. With **-d2**, for example, the sequence of directors or routers that process an address is output. If there's a message stuck on Exim's spool, you can force a delivery with debugging turned on by a command of the form

```
exim -d9 -M <message-id>
```

One specific problem that has shown up on some sites is the inability to do local deliveries into a single shared mailbox directory that does not have the 'sticky bit' set on it. By default, Exim tries to create a lock file before writing to a mailbox file, and if it cannot create the lock file, the delivery is deferred. You can get round this either by setting the 'sticky bit' on the directory, or by setting a specific group for local deliveries and allowing that group to create files in the directory (see the comments above the **local_delivery** transport in the default configuration file). Another approach is to configure Exim not to use lock files, but just to rely on **fcntl()** locking instead. However, you should do this only if all user agents also use **fcntl()** locking. For further discussion of locking issues, see chapter 15.

One thing that cannot be tested on a system that is already running a mailer is the receipt of incoming SMTP mail on the standard SMTP port. However, the **-oX** option can be used to run an Exim daemon that listens on some other port, or **inetd** can be used to do this.

Testing a new version on a system that is already running Exim can most easily be done by building a binary with a different CONFIGURE_FILE setting. From within the runtime configuration, all other file and directory names that Exim uses can be altered, in order to keep it entirely clear of the production version.

4.15 Switching Exim on

Building and installing Exim does not of itself put it in general use. The name by which the system message transfer agent is called by mail user agents is **/usr/lib/sendmail**, and it is necessary to make this name point to the **exim** binary in order to get them to use it. This is normally done by renaming any existing file and making **/usr/lib/sendmail** a symbolic link to the **exim** binary. It is then necessary to stop and restart the mailer daemon, if one is running.

4.16 Exim on heavily loaded hosts

If you are running Exim on a heavily loaded host you should consider installing a current release of **bind** (from **http://www.isc.org**) as caching nameserver, either locally or on a nearby host with fast communications. You should also consider enabling Exim's **split_spool_directory** if you expect to have large numbers of messages awaiting delivery.

4.17 Stopping Exim on Solaris 2

The standard command for stopping the mailer daemon on Solaris 2 is

```
  /etc/init.d/sendmail stop
```

If **/usr/lib/sendmail** has been turned into a symbolic link, this script fails to stop Exim because it uses the command **ps -e** and greps the output for the text ‘sendmail’; this is not present because the actual program name (that is, ‘exim’) is given by the **ps** command with these options. A fix that appears to work on Solaris 2.5 and above is to change the script so that the **ps** command reads

```
  ps -e -o pid,comm
```

which causes the name by which the daemon was started (that is, **/usr/lib/sendmail**) to be output. However, this fails if the daemon has been restarted with SIGHUP because Exim restarts itself using the real file name. A better solution is to replace the line that finds the process id with something like

```
  pid=‘cat /var/spool/exim/exim-daemon.pid‘
```

to obtain the daemon's pid directly from the file that Exim saves it in. See the description of the **-bd** option for details of where Exim writes the daemon's process id file.

5. The Exim command line

Exim's command line takes the standard Unix form of a sequence of options, each starting with a hyphen character, followed by a number of arguments. The options are compatible with the main options of Sendmail, and there are also some additional options, some of which are compatible with Smail 3. Certain combinations of options do not make sense, and provoke an error if used. The form of the arguments depends on which options are set.

5.1 Setting options by program name

If Exim is called under the name **mailq**, it behaves as if the option **-bp** were present before any other options. This is for compatibility with some systems that contain a command of that name in one of the standard libraries, symbolically linked to **/usr/lib/sendmail**.

If Exim is called under the name **rsmtp** it behaves as if the option **-bS** were present before any other options, for compatibility with **smail**. The **-bS** option is used for reading in a number of messages in batched SMTP format.

If Exim is called under the name **rmail** it behaves as if the **-i** and **-oee** options were present before any other options, for compatibility with **smail**. The name **rmail** is used as an interface by some UUCP systems.

If Exim is called under the name **runq** it behaves as if the option **-q** were present before any other options, for compatibility with **smail**. The **-q** option causes a single queue-runner process to be started.

If Exim is called under the name **newaliases** it behaves as if the option **-bi** were present before any other options, for compatibility with Sendmail. This option is used for rebuilding Sendmail's alias file. Exim does not have the concept of a single alias file, but can be configured to run a given command if called with the **-bi** option.

5.2 Trusted and admin users

Some Exim options are available only to *trusted users* and others are available only to *admin users*.

- A trusted user is root or the Exim user (if defined) or any user listed in the **trusted_users** configuration option, or any user for whom the currently set group is one of those listed in the **trusted_groups** configuration option.

  Trusted users are permitted to use the **-f** option or a leading 'From ' line to specify the envelope sender of a message that is passed to Exim through the local interface (see the **-bm** and **-f** options below). They may also specify a host name, host address, interface address, protocol name, and ident value. Thus they are able to insert messages into Exim's queue locally that have the characteristics of messages received from a remote host.

  From: headers are not checked to see if **Sender:** is needed when the caller is trusted.

- An admin user is root or the Exim user (if defined) or any user that is a member of the Exim group (if defined). The current group does not have to be the Exim group. Admin users are permitted to operate on messages in the queue, for example, to force delivery failures. It is also necessary to be an admin user in order to see the full information provided by the Exim monitor.

  By default, the use of the **-M**, **-q**, and **-R** options to cause Exim to attempt delivery of messages on its queue is restricted to admin users. However, this restriction can be relaxed by setting the **prod_requires_admin** option false (that is, specifying **no_prod_requires_admin**).

  Similarly, the use of the **-bp** option to list all the messages in the queue is restricted to admin users unless **queue_list_requires_admin** is set false.

5.3 Command line options

The command options are described in alphabetical order below.

-- This is a pseudo-option whose only purpose is to terminate the options and therefore to cause subsequent command line items to be treated as arguments rather than options, even if they begin with hyphens.

-bd Run Exim as a daemon, awaiting incoming SMTP connections. This option can be used only by an admin user. If either of the **-d** or **-dm** options are set, the daemon does not disconnect from the controlling terminal. By default, Exim listens for incoming connections on all the host's interfaces, but it can be restricted to specific interfaces by setting the **local_interfaces** option in the configuration file. The standard SMTP port is used, but this can be varied by means of the **daemon_smtp_port** configuration option or the **-oX** command line option. Most commonly, the **-bd** option is combined with the **-q**<*time*> option, to cause periodic queue runs to happen as well.

The process id of a daemon that is both listening on the standard SMTP port and periodically starting queue runners is written to a file called **exim-daemon.pid** in Exim's spool directory. If a non-standard port is used, the file name is **exim-daemon.**<*port-number*>**.pid**. If a daemon is run with only one of **-bd** or **-q**<*time*>, then that option is added on to the end of the file name, allowing sites that run two separate daemons to distinguish them.

It is possible to change the directory in which these pid files are written by changing the setting of PID_FILE_PATH in **Local/Makefile**. The files are written while Exim is still running as root. Further details are given in the comments in **src/EDITME**.

The SIGHUP signal can be used to cause the daemon to re-exec itself. This should be done whenever Exim's configuration file is changed, or a new version of Exim is installed. It is not necessary to do this when other files (for example, alias files) are changed.

-bF <*filename*>
This option is the same as **-bf** except that it assumes that the filter being tested is a system filter. The additional commands that are available only in system filters are recognized.

-bf <*filename*>
Run Exim in filter testing mode; the file is the filter file to be tested, and a test message must be supplied on the standard input. If there are no message-dependent tests in the filter, an empty file can be supplied. If a system filter file is being tested, **-bF** should be used instead of **-bf**. If the test file does not begin with the special line

```
# Exim filter
```

then it is taken to be a normal **.forward** file, and is tested for validity under that interpretation. The result of this command, provided no errors are detected, is a list of the actions that Exim would try to take if presented with the message for real. More details of filter testing are given in the separate document entitled *Exim's User interface to mail filtering*.

When testing a filter file, the envelope sender can be set by the **-f** option, or by a 'From ' line at the start of the test message. Various parameters that would normally be taken from the envelope recipient address of the message can be set by means of additional command line options. These are:

-bfd <*domain*>	default is the qualify domain
-bfl <*local_part*>	default is the logged in user
-bfp <*local_part_prefix*>	default is null
-bfs <*local_part_suffix*>	default is null

The local part should always be set to the incoming address with any prefix or suffix stripped, because that is how it appears when a message is actually being delivered.

-bh <*IP address*>
This option runs a fake SMTP session as if from the given IP address, using the standard input and output. Comments as to what is going on are written to the standard error file.

These include lines beginning with 'LOG' for anything that would have been logged. This facility is for testing configuration options for blocking hosts and/or senders and for checking on relaying control. Messages supplied during the testing session are discarded, and nothing is written to any of the real log files. There may be pauses when DNS (and other) lookups are taking place, and of course these may time out. The **-oMi** option can be used to specify a specific IP interface if this is important.

-bi Sendmail interprets the **-bi** option as a request to rebuild its alias file. Exim does not have the concept of a single alias file, and so it cannot mimic this behaviour. However, calls to **/usr/lib/sendmail -bi** tend to appear in various scripts such as NIS make files, so the option must be recognized.

If **-bi** is encountered, the command specified by the **bi_command** configuration option is run, under the uid and gid of the caller of Exim. If the **-oA** option is used, its value is passed to the command as an argument. The command set by **bi_command** may not contain arguments. The command can use the **exim_dbmbuild** utility, or some other means, to rebuild alias files if this is required. If the **bi_command** option is not set, then calling Exim with **-bi** is a no-op.

-bm Accept an incoming, locally-generated message on the current input, and deliver it to the addresses given as the command arguments (except when **-t** is also given – see below). Each argument can be a comma-separated list of RFC 822 addresses. This is the default option, and is assumed if no other conflicting option is present.

The format of the message must be as defined in RFC 822, except that, for compatibility with **sendmail** and **smail**, a line in one of the forms

```
From sender Fri Jan  5 12:55 GMT 1997
From sender Fri, 5 Jan 97 12:55:01
```

(with the weekday optional, and possibly with additional text after the date) is permitted to appear at the start of the message. There appears to be no authoritative specification of the format of this line. Exim recognizes it by matching against the regular expression defined by the **uucp_from_pattern** option, which can be changed if necessary. The specified sender is treated as if it were given as the argument to the **-f** option, but if a **-f** option is also present, its argument is used in preference to the address taken from the message. The caller of Exim must be a trusted user for the sender of a message to be set in this way.

-bp List the contents of the mail queue on the standard output. If the **-bp** option is followed by a list of message ids, then just those messages are listed. By default, this option lists only those messages submitted by the calling user unless the caller is an admin user. The **queue_list_requires_admin** option can be set false to allow any user to see the entire queue.

Each message on the queue is displayed as in the following example:

```
25m  2.9K 0t5C6f-0000c8-00 <alice@wonderland.fict.book>
          red.king@looking-glass.fict.book
          <other addresses>
```

The first line contains the length of time the message has been on the queue (in this case 25 minutes), the size of the message (2.9K), the unique local identifier for the message, and the message sender, as contained in the envelope. If the message is a delivery error message, the sender address is empty, and appears as '<>'. If the message is frozen (attempts to deliver it are suspended) then the text '*** frozen ***' is displayed at the end of this line.

The recipients of the message (taken from the envelope, not the headers) are displayed on subsequent lines. Those addresses to which the message has already been delivered are marked with the letter D. If an original address gets expanded into several addresses via an alias or forward file, the original is displayed with a D only when deliveries for all of its child addresses are complete.

-bpa This option operates like **-bp**, but in addition it shows delivered addresses that were generated from the original top level address(es) in each message by alias or forwarding operations. These addresses are flagged with '+D' instead of just 'D'.

-bpr This option operates like **-bp**, but the output is not sorted into chronological order of message arrival. This can speed it up when there are lots of messages on the queue, and is particularly useful if the output is going to be post-processed in a way that doesn't need the sorting.

-bpra This option is a combination of **-bpr** and **-bpa**.

-bpru This option is a combination of **-bpr** and **-bpu**.

-bpu This option operates like **-bp** but shows only undelivered top-level addresses for each message displayed. Addresses generated by aliasing or forwarding are not shown, unless the message was deferred after processing by a director with the **one_time** option set.

-bP If this option is given with no arguments, it causes the values of all Exim's main configuration options to be written to the standard output. The values of one or more specific options can be requested by giving their names as arguments, for example:

```
exim -bP qualify_domain local_domains
```

If **configure_file** is given, the name of the runtime configuration file is output. If **log_file_path** or **pid_file_path** are given, the names of the directories where log files and daemon pid files are written are output, respectively. If these values are unset, log files are written in a sub-directory of the spool directory called **log**, and pid files are written directly into the spool directory.

If one of the words **director**, **router**, or **transport** is given, followed by the name of an appropriate driver instance, the option settings for that driver are output. For example:

```
exim -bP transport local_delivery
```

The generic driver options are output first, followed by the driver's private options. A list of the names of drivers of a particular type can be obtained by using one of the words **director_list**, **router_list**, or **transport_list**, and a complete list of all drivers with their option settings can be obtained by using **directors**, **routers**, or **transports**.

-brt This option is for testing retry rules, and it must be followed by up to three arguments. It causes Exim to look for a retry rule that matches the values and to write it to the standard output. For example:

```
exim -brt bach.comp.mus
Retry rule: *.comp.mus  F,2h,15m; FG,4d,30m;
```

See chapter 31 for a description of Exim's retry rules. The first argument, which is required, can be a complete address in the form **local_part@domain**, or it can be just a domain name. The second argument is an optional second domain name; if no retry rule is found for the first argument, the second is tried. This ties in with Exim's behaviour when looking for retry rules for remote hosts – if no rule is found that matches the host, one that matches the mail domain is sought. The final argument is the name of a specific delivery error, as used in setting up retry rules, for example 'quota_3d'.

-brw This option is for testing address rewriting rules, and it must be followed by a single argument, consisting of either a local part without a domain, or a complete address with a fully qualified domain. Exim outputs how this address would be rewritten for each possible place it might appear. See chapter 32 for further details.

-bS This option is used for batched SMTP input, where messages have been received from some external source by an alternative transport mechanism. It causes Exim to accept one or more messages by reading SMTP on the standard input, but to generate no responses. If any error is encountered reports are written to the standard output and error streams, and Exim gives up immediately.

If the caller is trusted, then the senders in the MAIL commands are believed; otherwise the sender is always the caller of Exim. Unqualified senders and receivers are not rejected (there seems little point) but instead just get qualified. Sender addresses are verified if **sender_verify** is set, unless **sender_verify_batch** is unset (which is the default). Receiver verification and administrative rejection is not done, even if configured. HELO and EHLO act as RSET; VRFY, EXPN, ETRN, HELP, and DEBUG act as NOOP; QUIT quits. The return code is 0 if no error was detected; it is 1 if one or more messages were accepted before the error was detected; otherwise it is 2. More details of input using batched SMTP are given in section 42.8.

-bs This option causes Exim to accept one or more messages by reading SMTP commands on the standard input, and producing SMTP replies on the standard output. Some user agents use this interface as a way of passing locally-generated messages to the MTA. The option can also be used to run Exim from **inetd**, as an alternative to using a listening daemon, in which case the standard input is the connected socket. Exim distinguishes between the two cases by attempting to read the IP address of the peer connected to the standard input. If it is not a socket, the call to **getpeername()** fails, and Exim assumes it is dealing with a local message.

If the caller of Exim is trusted, then the senders of messages are taken from the SMTP MAIL commands. Otherwise the content of these commands is ignored and the sender is set up as the calling user.

-bt Run in address testing mode, in which each argument is taken as an address to be tested. The results are written to the standard output. If no arguments are given, Exim runs in an interactive manner, prompting with a right angle bracket for addresses to be tested. Each address is handled as if it were the recipient address of a message and passed to the appropriate directors or routers (compare the **-bv** option); the result is written to the standard output. The return code is 2 if any address failed outright; it is 1 if no address failed outright but at least one could not be resolved for some reason. Return code 0 is given only when all addresses succeed.

If any of the directors or routers in the configuration makes any tests on the sender address of a message, then you should use the **-f** option to set an appropriate sender when running **-bt** tests. Without it, the sender is assumed to be the calling user at the default qualifying domain.

-bV Write the current version number, compilation number, and compilation date of the **exim** binary to the standard output.

-bv Verify the addresses that are given as the arguments to the command, and write the results to the standard output. Verification differs from address testing (the **-bt** option) in that directors and routers that have **no_verify** set are skipped, and if the address is accepted by a director or router that has **fail_verify** set, verification fails. This is the same logic that is used when verifying addresses of incoming messages (see chapter 39). The address is verified as a recipient if **-bv** is used; to verify as for a sender address, **-bvs** should be used.

If the **-v** (or **-d**) option is not set, the output consists of a single line for each address, stating whether it was verified or not, and giving a reason in the latter case. Otherwise, more details are given of how the address has been handled, and in the case of aliases or forwarding, the generated addresses are also considered.

The return code is 2 if any address failed outright; it is 1 if no address failed outright but at least one could not be resolved for some reason. Return code 0 is given only when all addresses succeed.

If any of the directors or routers in the configuration makes any tests on the sender address of a message, then you should use the **-f** option to set an appropriate sender when running **-bv** tests. Without it, the sender is assumed to be the calling user at the default qualifying domain.

-bvs This option acts like **-bv**, but verifies the address as a sender rather than a recipient address. This affects any rewriting and qualification that might happen.

-C *<filename>*
Read the runtime configuration from the given file instead of from the default file specified by

the CONFIGURE_FILE compile-time setting. When this option is used by an unprivileged caller and the file name given is different from the compiled-in name, Exim gives up its root privilege immediately, and runs with the real and effective uid and gid set to those of the caller, to avoid any security exposure. It does not do this if the caller is root or the exim user. The facility is useful for ensuring that configuration files are syntactically correct, but cannot be used for test deliveries, unless the caller is privileged, or unless it's an exotic configuration that does not require privilege. No check is made on the owner or group of the file specified by this option.

-D<*macro*>=<*value*>
This option can be used to override macro definitions in the configuration file (see section 7.2). However, like **-C**, if it is used by an unprivileged caller, it causes Exim to give up its root privilege. This option may be repeated up to 10 times on a command line.

-d<*number*>
Set a debug level, causing debugging information to be written to the standard error file. White space between **-d** and the number is optional. If no number is given, 1 is assumed, and the higher the number, the more output is produced. A value of zero turns debugging output off and is the default. A value of 9 gives the maximum amount of general information, 10 gives in addition details of the interpretation of filter files, and 11 or higher also turns on the debugging option for DNS lookups.

-df If this option is set and STDERR_FILE was defined when Exim was built, debugging information is written to the file defined by that variable instead of to the standard error file. This option provides a way of obtaining debugging information when Exim is run from **inetd**.

-dm This option causes information about memory allocation and freeing operations to be written to the standard error file.

-dropcr At least one MUA (dtmail) that calls an MTA via the command line is broken in that it terminates each line with CRLF, instead of just LF, which is the usual Unix convention, and although this bug has been admitted, it apparently won't get fixed. There is also some UUCP software which leaves CR at the ends of lines in messages. As a slight pander to these programs, the **-dropcr** option causes Exim to drop *all* CR characters in an incoming non-SMTP message.

-E This option specifies that an incoming message is a locally-generated delivery failure report. It is used internally by Exim when handling delivery failures and is not intended for external use. Its only effect is to stop Exim generating certain messages to the mailmaster, as otherwise message cascades could occur in some situations. As part of the same option, a message id may follow the characters **-E**. If it does, the log entry for the receipt of the new message contains the id, following 'R=', as a cross-reference.

-e*x* There are a number of **sendmail** options starting with **-oe** which seem to be called by various programs without the leading **o** in the option. For example, the **vacation** program uses **-eq**. Exim treats all options of the form **-e***x* as synonymous with the corresponding **-oe***x* options.

-F <*string*>
Set the sender's full name for use when a locally-generated message is being accepted. In the absence of this option, the user's *gecos* entry from the password file is used. As users are generally permitted to alter their *gecos* entries, no security considerations are involved. White space between **-F** and the <*string*> is optional.

-f <*address*>
Set the address of the sender of a locally-generated message. This option can normally be used only by root or the Exim user or by one of the configured trusted users. However, anyone may use it when testing a filter file with **-bf** or when testing or verifying addresses using the **-bt** or **-bv** options. In other cases, the sender of a local message is always set up as the user who ran the **exim** command, and **-f** is ignored, with one exception. If the special setting **-f <>** is used by an untrusted user, it does not affect the sender for the purposes of

managing the **Sender:** and **From:** headers, but it does have the effect of causing any SMTP transmissions to be sent out with

```
MAIL FROM:<>
```

and local deliveries to contain

```
Return-path: <>
```

when configured to contain **Return-path:** headers. The filtering code treats such a message as an error message, and won't generate messages as a result of reading it.

White space between **-f** and the <*string*> is optional. The sender of a locally-generated message can also be set by an initial 'From ' line in the message – see the description of **-bm** above, but if **-f** is also present, it overrides 'From '.

-h <*number*>
This option is accepted for compatibility with **sendmail**, but at present has no effect. (In **sendmail** it overrides the 'hop count' obtained by counting **Received:** headers.)

-i This option, which has the same effect as **-oi**, specifies that a dot on a line by itself should not terminate an incoming, non-SMTP message. I can find no documentation for this option in Solaris 2.4 **sendmail**, but the **mailx** command in Solaris 2.4 uses it.

-M The arguments are interpreted as a list of message ids, and Exim runs a delivery attempt on each message in turn. If any of the messages are frozen, they are automatically thawed before the delivery attempt. Retry hints for any of the addresses are overridden – Exim tries to deliver even if the normal retry time has not yet been reached. This option requires the caller to be an admin user. However, there is an option called **prod_requires_admin** which can be set false to relax this restriction (and also the same requirement for the **-q** and **-R** options).

-Mar <*message id*> <*address*> <*address*> **...**
The first argument must be a message id, and the remaining ones must be email addresses. Exim adds the addresses to the list of recipients of the message ('ar' for 'add recipients'). However, if the message is active (in the middle of a delivery attempt), its status is not altered. This option can be used only by an admin user.

-MC <*transport*> <*hostname*> <*sequence number*> <*message id*>
This option is not intended for use by external callers. It is used internally by Exim to invoke another instance of itself to deliver a waiting message using an existing SMTP channel, which is passed as the standard input and output. Details are given in chapter 42. This must be the final option, and the caller must be root or the Exim user in order to use it.

-MCQ <*process id*>
This option is not intended for use by external callers. It is used internally by Exim in conjunction with **-MC** option to pass on the process id of the queue runner which initiated the original delivery, if there was one.

-MCS This option is not intended for use by external callers. It is used internally by Exim in conjunction with **-MC** option, and passes on the fact that the SMTP SIZE option should be used on messages delivered down the existing channel.

-Mc The arguments are interpreted as a list of message ids, and Exim runs a delivery attempt on each message in turn, but unlike the **-M** option, it does check for retry hints, and respects any that are found. This option is not very useful to external callers (except for testing). It is provided for internal use by Exim when it needs to re-invoke itself in order to regain root privilege for a delivery (see chapter 49).

-Meb <*message id*>
This runs, under /bin/sh, the command defined in the shell variable VISUAL or, if that is not defined, EDITOR or, if that is not defined, the command **vi**, on a copy of the spool file containing the body of message ('eb' for 'edit body'). If the editor exits normally, then the result of editing replaces the spool file. The message is locked during this process, so no

delivery attempts can occur. Note that the first line of the spool file is its own name; care should be taken not to disturb this. The thinking behind providing this feature is that an administrator who has had to mess around with the addresses to get a message delivered might want to add some comment at the start of the message text. This option can be used only by an admin user.

-Mes <*message id*> <*address*>
There must be exactly two arguments. The first argument must be a message id, and the second one an email address. Exim changes the sender address in the message to the given address, which must be a fully qualified address or '<>' ('es' for 'edit sender'). However, if the message is active (in the middle of a delivery attempt), its status is not altered. This option can be used only by an admin user.

-Mf The arguments are interpreted as a list of message ids, and each message is marked 'frozen'. This prevents any delivery attempts taking place until the message is 'thawed', either manually or as a result of the **auto_thaw** configuration option. However, if any of the messages are active (in the middle of a delivery attempt), their status is not altered. This option can be used only by an admin user.

-Mg The arguments are interpreted as a list of message ids, and Exim gives up trying to deliver those messages. A delivery error message is sent, containing the text 'cancelled by administrator'. However, if any of the messages are active, their status is not altered. This option can be used only by an admin user.

-Mmad <*message id*>
Exim marks all the recipient addresses in the message as already delivered ('mad' for 'mark all delivered'). However, if the message is active (in the middle of a delivery attempt), its status is not altered. This option can be used only by an admin user.

-Mmd <*message id*> <*address*> <*address*> **...**
The first argument must be a message id, and the remaining ones must be email addresses. Exim marks the given addresses as already delivered ('md' for 'mark delivered'). However, if the message is active (in the middle of a delivery attempt), its status is not altered. This option can be used only by an admin user.

-Mrm The arguments are interpreted as a list of message ids, and each message is completely removed from Exim's queue, and forgotten. However, if any of the messages are active, their status is not altered. This option can be used only by an admin user or by the user who originally caused the message to be placed on the queue.

-Mt The arguments are interpreted as a list of message ids, and each message that was 'frozen' is now 'thawed', so that delivery attempts can resume. However, if any of the messages are active, their status is not altered. This option can be used only by an admin user.

-Mvb <*message id*>
The contents of the message body (-D) spool file are written to the standard output. This option can be used only by an admin user.

-Mvh <*message id*>
The contents of the message headers (-H) spool file are written to the standard output. This option can be used only by an admin user.

-Mvl <*message id*>
The contents of the message log spool file are written to the standard output. This option can be used only by an admin user.

-m This is apparently a synonym for **-om** that is accepted by **sendmail**, so Exim treats it that way too.

-N This is a debugging option that inhibits delivery of a message at the transport level. It implies at least **-d1**. Exim goes through many of the motions of delivery – it just doesn't actually transport the message, but instead behaves as if it had successfully done so. However, it does not make any updates to the retry database, and the log entries for deliveries are flagged with '*>' rather than '=>'. Only root or the exim user are allowed to use **-N** with **-bd**, **-q**, **-R** or **-M**. In other words, an ordinary user can use it only when supplying an incoming message to which it will apply.

-oA <*file name*>
This option is used by Sendmail in conjunction with **-bi** to specify an alternative alias file name. Exim handles **-bi** differently; see the description above.

-oB <*n*> This is a debugging option which limits the maximum number of multiple SMTP deliveries down one channel to <*n*>, overriding the value set in the **smtp** transport. If <*n*> is omitted, the limit is set to 1 (no batching).

-odb This option applies to all modes in which Exim accepts incoming messages, including the listening daemon. It requests 'background' delivery of such messages, which means that the accepting process automatically starts another delivery process for each message received. Exim does not wait for such processes to complete (it can take some time to perform SMTP deliveries). This is the default action if none of the **-od** options are present.

-odf This option (compatible with **smail**) requests 'foreground' (synchronous) delivery when Exim has accepted a locally-generated message. For the daemon it is exactly the same as **-odb**. For a single message received on the standard input, if the protection regime permits it (see chapter 49), Exim converts the reception process into a delivery process. In other cases, it creates a new delivery process, and then waits for it to complete before proceeding.

-odi This option is synonymous with **-odf**. It is provided for compatibility with **sendmail**.

-odq This option applies to all modes in which Exim accepts incoming messages, including the listening daemon. It specifies that the accepting process should not automatically start a delivery attempt for each message received. Messages are placed on the queue, and remain there until a subsequent queue-running process encounters them. The **queue_only** configuration option has the same effect.

-odqr This option applies to all modes in which Exim accepts incoming messages, including the listening daemon. It causes Exim to process local addresses when a message is received, but not even to try routing remote addresses. Contrast with **-odqs** below, which does the routing, but not the delivery. The remote addresses will be picked up by the next queue runner. The **queue_remote** configuration option has the same effect for specific domains.

-odqs This option is a hybrid between **-odb** and **-odq**. A delivery process is started for each incoming message, the addresses are all processed, and local deliveries are done in the normal way. However, if any SMTP deliveries are required, they are not done at this time. Such messages remain on the queue until a subsequent queue-running process encounters them. Because routing was done, Exim knows which messages are waiting for which hosts, and so a number of messages for the same host will get sent in a single SMTP connection. The **queue_smtp** configuration option has the same effect for specific domains. See also the **-qq** option.

-oee If an error is detected while a non-SMTP message is being received (for example, a malformed address), the error is reported to the sender in a mail message. Provided the message is successfully sent, Exim exits with a return code of zero. If not, the return code is 2 if the error was that the message had no recipients, and 1 otherwise. This option is the default if Exim is called as **rmail**.

-oem This is the same as **-oee**, except that Exim always exits with a non-zero return code, whether or not the error message was successfully sent. This is the default option, unless Exim is called as **rmail**.

-oep If an error is detected while a non-SMTP message is being received, the error is reported by writing a message to the standard error file (stderr).

-oeq This option is supported for compatibility with **sendmail**, but has the same effect as **-oep**.

-oew This option is supported for compatibility with **sendmail**, but has the same effect as **-oem**.

-oi This option, which has the same effect as **-i**, specifies that a dot on a line by itself should not terminate an incoming, non-SMTP message. This is the default if Exim is called as **rmail**.

-oMa <*host address*>
This option sets the sender host address value, and can be used only by a trusted caller, except in conjunction with the **-bh** option. A real incoming connection overrides the address set by **-oMa**. The value is used in log entries and can appear in **Received:** headers. The option is intended for use when handing to Exim messages received by other means, either via the command line or by using the **-bs** option. If **-oMt** is set then **-oMa** should normally be set as well.

-oMi <*interface address*>
This option sets the IP interface address value, and can be used only by a trusted caller, except in conjunction with the **-bh** option. A real incoming connection overrides the address set by **-oMi**. The option is intended for use when handing to Exim messages received by other means, either via the command line or by using the **-bs** option.

-oMr <*protocol name*>
This option sets the received protocol value, and can be used only by a trusted caller, except in conjunction with the **-bh** option. The value is used in log entries and can appear in **Received:** headers. The option is intended for use when handing to Exim messages received by other means. It applies only to non-SMTP and batched SMTP input.

-oMs <*host name*>
This option sets the sender host name value, and can be used only by a trusted caller, except in conjunction with the **-bh** option. The value is used in log entries and can appear in **Received:** headers. The option is intended for use when handing to Exim messages received by other means.

-oMt <*ident string*>
This option sets the sender ident value, and can be used only by a trusted caller, except in conjunction with the **-bh** option. The value is used in log entries and can appear in **Received:** headers. The default setting for local callers is the login id of the calling process. This can be overridden by supplying an empty argument. The option is intended for use when handing to Exim messages received by other means.

-om In **sendmail**, this option means ‘me too’, indicating that the sender of a message should receive a copy of the message if the sender appears in an alias expansion. Exim always does this, so the option does nothing.

-or <*time*>
This option sets a timeout value for incoming non-SMTP messages. If it is not set, Exim will wait forever for the standard input. The value can also be set using the **accept_timeout** configuration variable. The format used for specifying times is described in section 7.7.

-ov This option has exactly the same effect as **-v**.

-oX <*number*>
This option is relevant only when the **-bd** option is also given. It overrides any setting of the **daemon_smtp_port** option, and specifies an alternative TCP/IP port number for the listening daemon. When used, the process number of the daemon is written to a file whose name is **exim-daemon.**<*number*>**.pid** in Exim’s spool directory or the directory specified by PID_FILE_PATH in **Local/Makefile**.

-pd This option applies when an embedded Perl interpreter is linked with Exim (see chapter 10). It overrides the setting of the **perl_at_start** option, forcing the starting of the interpreter to be delayed until it is needed.

-ps This option applies when an embedded Perl interpreter is linked with Exim (see chapter 10). It overrides the setting of the **perl_at_start** option, forcing the starting of the interpreter to occur as soon as Exim is started.

-q If the **-q** option is not followed by a time value, it requests a single queue run operation. This option requires the caller to be an admin user. However, there is an option called **prod_requires_admin** which can be set false to relax this restriction (and also the same requirement for the **-M** and **R** options).

Exim starts up a delivery process for each (inactive) message on the queue in turn, and waits for it to finish before starting the next one. If the delivery process spawns other processes to deliver other messages down passed SMTP connections, the queue runner waits for these to finish before proceeding. When all the queued messages have been considered, the original process terminates. In other words, a single pass is made over the waiting mail. Use **-q** with a time (see below) if you want this to be repeated periodically.

Exim processes the waiting messages in an unpredictable order. It isn't very random, but it is likely to be different each time, which is all that matters. If one particular message screws up a remote MTA, other messages to the same MTA have a chance of getting through if they get tried first.

However, it is possible to cause the messages to be processed in lexical message id order, which is essentially the order in which they arrived, and to start this operation at a particular point by following the **-q** option with a starting message id. For example:

```
exim -q 0t5C6f-0000c8-00
```

This causes Exim to skip any messages whose ids are lexically less than the given id. A second id can also be given to stop the queue run before the end. See also **-R** and the **queue_run_in_order** option.

-q *<time>*
This version of the **-q** option (which again can be run only by an admin user) causes Exim to run as a daemon, starting a queue-running process at intervals specified by the given time value (whose format is described in section 7.7). This form of the **-q** option is commonly combined with the **-bd** option, in which case a single daemon process handles both functions. A common way of starting up a combined daemon at system boot time is to use a command such as

```
/opt/exim/bin/exim -bd -q30m
```

Such a daemon listens for incoming SMTP calls, and also fires up a queue-runner process every 30 minutes. The process id of such a daemon is written to a file called **exim-daemon.pid** in Exim's spool directory, unless the **-oX** option has been used, in which case the file is called **exim-daemon.**<*port-number*>**.pid**. The location of the pid file can be changed by defining PID_FILE_PATH in **Local/Makefile**. If a daemon is started without **-bd** then the **-q** option used to start it is added to the pid file name.

-qf This option operates like **-q**, and may appear with or without a following time. The difference is that a delivery attempt is forced for each non-frozen message, whereas with **-q** only those non-frozen addresses that have passed their retry times are tried.

-qff This option operates like **-qf** and may appear with or without a following time. The difference is that any frozen messages are automatically thawed, and delivery is attempted for them.

-qfl This option operates like **-ql**, and may appear with or without a following time. The difference is that a delivery attempt is forced for the local addresses in each non-frozen message, whereas with **-ql** only those non-frozen local addresses that have passed their retry times are tried.

-qffl This option operates like **-qfl** and may appear with or without a following time. The difference is that any frozen messages are automatically thawed, and delivery is attempted for any local addresses in them.

-ql This option operates like **-q**, and may appear with or without a following time. The difference is that only local addresses (those with domains that match **local_domains**) are considered for delivery. Note that **-ql** cannot detect apparently remote addresses that actually turn out to be local when their domains get fully qualified.

-qq... If any command line option starting with **-q** is specified with an additional **q** (for example, **-qqf**) then all the resulting queue runs are done in two stages. In the first stage, the queue is scanned as if the **queue_smtp_domains** option matched every domain. This causes remote addresses to be routed, but no transportation to be done. The database that remembers which messages are waiting for specific hosts is updated, as if delivery to those hosts had been deferred. After this is complete, a second, normal queue scan happens, and normal directing, routing, and delivery takes place. Messages which are routed to the same host should mostly be delivered down a single SMTP connection because of the hints that were set up during the first queue scan. This option may be useful for hosts that are connected to the Internet intermittently.

-qR *<string>*
This option is synonymous with **-R**. It is provided for **sendmail** compatibility.

-qRf *<string>*
This option is synonymous with **-Rf**.

-R *<string>*
The white space between **-R** and the string is optional. This option is similar to **-q** with no time value, except that, when scanning the messages on the queue, Exim processes only those that are not frozen and have at least one undelivered address containing the given string, which is checked in a case-independent way. However, once a message is selected, all its addresses are processed. For the first message containing a matching address, Exim overrides any retry information and forces a delivery attempt. This makes it straightforward to initiate delivery of all messages to a given domain after a host has been down for some time. When the SMTP command ETRN is permitted (see the **smtp_etrn** options), its default effect is to run Exim with the **-R** option.

-Rf *<string>*
This option acts like **-R**, but forces a delivery for every matching non-frozen message, not just the first one. White space is required between **-Rf** and the string.

-Rff *<string>*
This option acts like **-Rf**, but also thaws any frozen messages it encounters.

-r This is a documented (for **sendmail**) obsolete alternative name for **-f**.

-t When Exim is receiving a locally-generated, non-SMTP message on the current input, the **-t** option causes the recipients of the message to be obtained from the **To:**, **Cc:**, and **Bcc:** headers in the message instead of from the command arguments. The addresses are extracted before any rewriting takes place.

If there are in fact any arguments, they specify addresses to which the message is *not* to be delivered. That is, the argument addresses are removed from the recipients list obtained from the headers. This is compatible with Smail 3 and in accordance with the documented behaviour of Sendmail. However, it has been reported that in some versions at least, Sendmail adds argument addresses to those obtained from the headers. Exim can be made to behave in this way by setting the option **extract_addresses_remove_arguments** false.

If a **Bcc:** header is present, it is removed from the message unless there is no **To:** or **Cc:** header, in which case a **Bcc:** header with no data is created, in accordance with RFC 822.

-v This option has exactly the same effect as **-d1**; it causes Exim to be ‘verbose’ and produce some output describing what it is doing on the standard error file. In particular, if an SMTP connection is made, the SMTP dialogue is shown.

-x AIX uses **-x** for a private purpose (‘mail from a local mail program has National Language Support extended characters in the body of the mail item’). It sets **-x** when calling the MTA from its **mail** command. Exim ignores this option.

6. File and database lookups

Exim can be configured to look up data in files or databases in a number of different circumstances. This chapter discusses some of the common features of the data lookup facilities; particular cases are covered in more detail in subsequent chapters.

Two different styles of data lookup are implemented:

- The *single-key* style requires the specification of a file in which to look, and a single key to search for.
- The *query* style accepts a generalized query, which may contain one or more keys.

The code for each lookup type is in a separate source file which is compiled and included in the binary of Exim only if the corresponding compile-time option is set. The default settings in **src/EDITME** are:

```
LOOKUP_DBM=yes
LOOKUP_LSEARCH=yes
```

which means that only linear searching and DBM lookups are included by default.

6.1 Single-key lookup types

The following single-key lookup types are implemented:

- **lsearch**: The given file is searched linearly for a line beginning with the single key, terminated by a colon or white space or the end of the line. White space between the key and the colon is permitted. The remainder of the line, with leading and trailing white space removed, is the data. This can be continued onto subsequent lines by starting them with any amount of white space, but only a single space character is included in the data at such a junction. If the data begins with a colon, then the key must be terminated by a colon, for example:

  ```
  baduser:  :fail:
  ```

  Empty lines and lines beginning with # are ignored, even if they occur in the middle of an item. This is the traditional textual format of alias files.

- **dbm**: Calls to DBM library functions are used to extract data from the given DBM file by looking up the record with the given key. The terminating binary zero is included in the key that is passed to the DBM library.

- **nis**: The given file is the name of a NIS map, and a NIS lookup is done with the given key, excluding the terminating binary zero. There is a variant called **nis0** which does include the terminating binary zero in the key. This is needed for Sun-style alias files. Exim does not recognize NIS aliases; the full map names must be used.

- **cdb**: The given file is searched as a Constant DataBase file, using the key string without the terminating binary zero. The cdb format is designed for indexed files that are read frequently and never updated, except by total re-creation. As such, it is particulary suitable for large files containing aliases or other indexed data referenced by an MTA. Information about cdb can be found at

  http://www.pobox.com/~djb/cdb.html

  The cdb distribution is not needed in order to build Exim with cdb support, as the code for reading cdb files is included directly in Exim itself. However, no means of building or testing cdb files is provided with Exim because these are available within the cdb distribution.

If '∗' is added to a single-key lookup type (for example, **lsearch∗**), then if the initial lookup fails, the key '∗' is looked up in the file to provide a default value. See also the section on partial matching below.

Alternatively, if '∗@' is added to a single-key lookup type (for example **dbm∗@**) then, if the initial lookup fails and the key contains an @ character, a second lookup is done with everything before the last @ replaced by ∗. This makes it possible to provide per-domain defaults in alias files that include the domains in the keys. If the second lookup fails, then '∗' is looked up.

There has been some confusion about the way **lsearch** lookups work, in particular in a domain or host list. An item in one of these lists may be a plain file name, or a file name preceded by a search type, and these behave differently. For a plain file name, for example

```
local_domains = /etc/local-mail-domains
```

each line of the file is treated as if it appeared as an item in the list, and negated items, wild cards, and regular expressions may be present. However, if an item is specified as an **lsearch** lookup, for example

```
local_domains = lsearch;/etc/local-mail-domains
```

then negated items, wild cards, and regular expressions may not be used, because **lsearch** is an indexed lookup method which, when given a key (the domain in the above example), yields a data value that corresponds to that key. The fact that the file is searched linearly does not make this kind of search any different from the other single-key lookup types, and an **lsearch** file can always be directly converted into one of the other types. Thus the keys in the file are literal strings and are not interpreted in any way.

6.2 Query-style lookup types

The following query-style lookup types are implemented:

- **nisplus**: This does a NIS+ lookup using a query that may contain any number of keys, and which can specify the name of the field to be returned. See section 6.7 below.
- **ldap**: This does an LDAP lookup using a query in the form of a URL. There is a variant called **ldapm** which permits values from multiple entries to be returned. See section 6.8 below.
- **dnsdb**: This does a DNS search for a TXT record whose domain name is the supplied query. The resulting data is the contents of the TXT record.
- **testdb**: This is a lookup type which is for use in debugging Exim. It is not likely to be useful in normal operation.

6.3 Use of data lookups

There are three different types of configuration item in which data lookups can be specified:

(1) Any string that is to be expanded may contain explicit lookup requests. String expansions are described in chapter 9.

(2) Lists of domains and other items can contain lookup requests as a way of avoiding excessively long linear lists. See section 7.12 for a full description.

(3) Some drivers can be configured directly to look up data in files.

In a string expansion, all the parameters of the lookup are specified explicitly, while for the other types there is always one implicit key involved. For example, the **local_domains** option contains a list of local domains; when it is being searched there is some domain name that is an implicit key.

This is not a problem for single-key lookups; the relevant file name is specified, and the key is implicit. For example, the list of local domains could be given as

```
local_domains = dbm;/local/domain/list
```

However, for query-style lookups the entire query has to be specified, and to do this, some means of including the implicit key is required. The special expansion variable **$key** is provided for this purpose. NIS+ could be used to look up local domains by a setting such as

```
local_domains = nisplus;[domain=$key],domains.org_dir
```

In cases where drivers can be configured to do lookups, there are always three alternative configuration options: **file** is used for single-key lookups, using an implicit key, and **query** or **queries** is specified for query-style lookups. In these cases the query is an expanded string, and the implicit key that would be used for **file** is always available as one of the normal expansion variables. The difference between **query** and **queries** is that in the latter case the string is treated as a colon-separated list of queries that are tried in order until one succeeds.

6.4 Temporary errors in lookups

Lookup functions can return temporary error codes if the lookup cannot be completed. (For example, a NIS or LDAP database might be unavailable.) When this occurs in a transport, director, or router, delivery of the message is deferred, as for any other temporary error. In other circumstances Exim may assume the lookup has failed, or may give up altogether. It is not advisable to use a lookup that might defer for critical options such as (to give an extreme example) **local_domains**.

6.5 Partial matching in single-key lookups

The normal operation of a single-key lookup is to search the file for an exact match with the given key. However, in a number of situations where domains are being looked up, it is possible to request partial matching. In this case, information in the file that has a key starting with '∗.' matches any domain that ends with the components that follow the fullstop. For example, if a key in a DBM file is

```
*.dates.fict.book
```

then this matches **2001.dates.fict.book** and **1984.dates.fict.book** when partial matching is enabled. It also matches **dates.fict.book**, if that key does not itself appear as a key in the file.

Partial matching is requested by adding the string 'partial-' to the front of the name of a search type, for example, **partial-dbm**. The subject key is first looked up verbatim; if that fails, then '∗.' is added at the start and it is looked up again. If that fails, then further lookups are tried with dot-separated components removed from the start, one-by-one, and '∗.' added on the front, until there are fewer than two non-∗ components left. For example, if subject key is **2250.dates.fict.book** then the following keys are looked up:

```
2250.dates.fict.book
*.2250.dates.fict.book
*.dates.fict.book
*.fict.book
```

As soon as one key in the sequence is successfully looked up, the lookup finishes.

The minimum number of non-∗ components can be adjusted by including a number before the hyphen in the search type. For example, **partial3-lsearch** specifies a minimum of three non-∗ components in the key. If 'partial0' is used, the original key gets shortened right down to the null string, and the final lookup is for '∗' on its own.

If the search type ends in '∗' or '∗@', then the search for an ultimate default happens after all partial lookups have failed. If 'partial0' is specified, adding '∗' to the search type has no effect, because the '∗' key is already included in the sequence of partial lookups.

The use of '∗' in lookup partial matching differs from its use as a wildcard in domain lists and the like. Partial matching works only in terms of dot-separated components; a key such as ***fict.book** in a database file is useless, because the asterisk in a partial matching subject key is always followed by a dot.

6.6 Lookup caching

Exim caches the most recent lookup result on a per-file basis for single-key lookup types, and keeps the relevant files open. In some types of configuration this can lead to many files being kept open for messages with many recipients. To avoid hitting the operating system limit on the number of simultaneously open files, Exim closes the least recently used file when it needs to open more files than its own internal limit, which can be changed via the **lookup_open_max** option. For query-style lookups, a single data cache per lookup type is kept. The files are closed and the caches flushed at strategic points during delivery – for example, after all directing and routing is complete.

6.7 More about NIS+

NIS+ queries consist of a NIS+ *indexed name* followed by an optional colon and field name. If this is given, the result of a successful query is the contents of the named field; otherwise the result consists of a concatenation of *field-name=field-value* pairs, separated by spaces. Empty values and values containing spaces are quoted. For example, the query

```
[name=mg1456],passwd.org_dir
```

might return the string

```
name=mg1456 passwd="" uid=999 gid=999 gcos="Martin Guerre"
home=/home/mg1456 shell=/bin/bash shadow=""
```

(split over two lines here to fit on the page), whereas

```
[name=mg1456],passwd.org_dir:gcos
```

would just return

```
Martin Guerre
```

with no quotes. A NIS+ lookup fails if NIS+ returns more than one table entry for the given indexed key.

6.8 More about LDAP

The include files and libraries needed to compile Exim with LDAP support can be obtained from

http://www.openldap.org

This code is a development of the original University of Michigan LDAP implementation. Other LDAP implementations exist; there is one from Netscape, and Solaris 7 contains inbuilt LDAP support. Unfortunately , though these are all compatible at the lookup function level, their error handling is different. For this reason it is necessary to set a compile-time variable when building Exim with LDAP, to indicate which LDAP library is in use. One of the following should appear in your **Local/Makefile**:

```
LDAP_LIB_TYPE=UMICHIGAN
LDAP_LIB_TYPE=NETSCAPE
LDAP_LIB_TYPE=SOLARIS7
```

If LDAP_LIB_TYPE is not set, Exim uses a heuristic to guess which of the University of Michigan or Netscape libraries is in use. It cannot distinguish the Solaris 7 library.

An LDAP query takes the form of a URL as defined in the Internet Draft **draft-ietf-asid-ldapv3-url-04.txt**. For example, in the configuration of an **aliasfile** director one might have these settings:

```
search_type = ldap
query = "ldap:///o=University%20of%20Cambridge,c=UK\
        ?mailbox?sub?(cn=$local_part)"
```

If an LDAP lookup finds an entry with no attributes, it behaves as if the entry did not exist.

The **ldap** lookup type generates an error if more than one entry matches the search filter, whereas **ldapm** permits this case. It is possible for multiple values, separated by newlines, to be returned for both **ldap** and **ldapm**, but in the former case you know that whatever values are returned all came from a single entry in the directory.

7. The Exim configuration file

Exim uses a single runtime configuration file which it reads when it is starting up. The name of the file is compiled into the binary for security reasons, and is specified by the CONFIGURE_FILE compilation option.

Some sites may wish to use the same Exim binary on different machines that share a file system, but to use different configuration files on each machine. If CONFIGURE_FILE_USE_NODE is defined in **Local/Makefile**, then Exim first looks for a file whose name is the configuration file name followed by a dot and the machine's node name, as obtained from the **uname()** function. If this file does not exist, the standard name is tried.

In some esoteric situations different versions of Exim may be run under different effective uids and the CONFIGURE_FILE_USE_EUID is defined to help with this. See the comments in **src/EDITME** for details.

The runtime configuration file must be owned by root or by the user that is specified at compile time by the EXIM_UID option, and it must not be world-writeable or group-writeable, unless its group is the one specified at compile time by the EXIM_GID option.

A one-off alternative configuration file can be specified by the **-C** command line option, but if this is done, Exim immediately gives up its root privilege, unless called by root or the Exim user, so this option is useful mainly for checking the syntax of configuration files before installing them. No owner or group checks are done on a configuration file specified by **-C**.

A default configuration file, which will work correctly in simple situations, is provided in the file **src/configure.default**. The installation process copies this into CONFIGURE_FILE if there is no previously-existing configuration file.

If a syntax error is detected while reading the configuration file, Exim writes a message on the standard error, and exists with a non-zero return code. The message is also written to the panic log.

7.1 Configuration file format

Exim's configuration file is in six parts, which must appear in the correct order in the file, separated by lines containing just the word 'end'. These parts contain:

- Main configuration settings.
- Configuration settings for the transport drivers. Transports define mechanisms for copying messages to destinations.
- Configuration settings for the director drivers. Directors process local addresses, that is, those with domains that match **local_domains**. These are typically (but not necessarily) delivered on the local host.
- Configuration settings for the router drivers. Routers process remote addresses, that is, those with domains that do not match **local_domains**.
- Retry rules, for use when a message cannot be immediately delivered.
- Address rewriting rules.

Blank lines in the file are ignored, and lines starting with a # character are treated as comments and are also ignored. Note that a # character other than at the beginning of a line is not treated specially, and does not introduce a comment. A convenient way to create a configuration file is to start from the default, which is supplied in **src/configure.default**, and add, delete, or change settings as required.

The retry and rewriting rules have their own syntax which is described in chapters 31 and 32. The earlier parts of the configuration file (whose setting are described in chapters 11 – 30) have some syntactic items in common, and these are described in sections 7.3 onwards. Before that, the simple macro facility is described.

7.2 Macros in the configuration file

If a line in the main part of the configuration (that is, before the first ‘end’ line) begins with an upper-case letter, it is taken as a macro definition, of the form

<*name*> = <*rest of line*>

The name must consist of letters, digits, and underscores, and need not all be in upper-case, though that is recommended. The rest of the line is the replacement text, and has leading and trailing white space removed. Quotes are not removed. If the line ends with a backslash character after trailing space is removed, then the next line is concatenated with it, with the backslash character and any leading space on the following line omitted. This continues for as long as lines end in backslash. Thus a replacement text can never end with a backslash character, but this doesn’t seem to be a serious limitation.

Once a macro is defined, all subsequent lines in the file are scanned for the macro name; if there are several macros, the line is scanned for each in turn, in the order in which they are defined. The replacement text is not re-scanned for the current macro, though it will be for subsequently defined macros. For this reason, a macro name may not contain the name of a previously defined macro as a substring. You could, for example, define

```
ABCD_XYZ = <<something>>
ABCD = <<something>>
```

but putting the definitions in the opposite order would provoke a configuration error.

As an example of macro usage, suppose you have lots of local domains, but they fall into three different categories. You could set up

```
LOCAL1 = domain1:\
         domain2
LOCAL2 = domain3:domain4
LOCAL3 = dbm;/list/of/other/domains

local_domains = LOCAL1:LOCAL2:LOCAL3
```

and then use the **domains** option on appropriate directors to handle each set of domains differently. This avoids having to list each domain in more than one place.

7.3 Common option syntax

For the main set of options and for driver options, each setting is on a line by itself, and starts with a name consisting of lower-case letters and underscores. Many options require a data value, and in these cases the name must be followed by an equals sign (with optional white space) and then the value. For example:

```
exim_user = exim
```

Options whose type is given as boolean are on/off switches that are not always followed by a data value. If the option name is specified on its own, the switch is turned on; if it is preceded by ‘no_’ or ‘not_’ then the switch is turned off. However, boolean options may be followed by an equals sign and one of the words ‘true’, ‘false’, ‘yes’, or ‘no’. For example:

```
sender_verify
no_smtp_verify
queue_only = true
```

The types of data that may be required by non-boolean options are described in the following sections.

7.4 Integer

If a numerical data item starts with the characters ‘0x’, the remainder of it is interpreted as a hexadecimal number. Otherwise, it is treated as octal if it starts with the digit 0, and decimal if not. If an integer value is followed by the letter K, it is multiplied by 1024; if it is followed by the letter M, it is multiplied by 1024x1024.

When the values of integer option settings are output, values which are an exact multiple of 1024 or 1024x1024 are printed using the letters K and M. The printing style is independent of the actual input format that was used.

7.5 Octal integer

The value of an option specified as an octal integer is always interpreted in octal, whether or not it starts with the digit zero. Such options are always output in octal.

7.6 Fixed point number

A fixed point number consists of a decimal integer, optionally followed by a decimal point and up to three further digits.

7.7 Time interval

A time interval is specified as a sequence of numbers, each followed by one of the following letters, with no intervening white space:

s	seconds
m	minutes
h	hours
d	days
w	weeks

For example, ‘3h50m’ specifies 3 hours and 50 minutes. The values of time intervals are output in the same format.

7.8 String

If a string data item does not start with a double-quote character, then it is taken as consisting of the remainder of the line, starting at the first character after any white space, with trailing white space characters removed, and with no interpretation of the characters therein.

If a string does start with a double-quote, then it continues to a closing double-quote, with the backslash character being interpreted as an escape character. If a backslash occurs at the end of an input line, the string is continued on the following line, with any leading white space being removed. Because Exim removes comment lines (those beginning with #) at an early stage, they can appear in the middle of a multi-line string.

The following two settings are equivalent:

```
trusted_users = "uucp:\
                 mail"
trusted_users = uucp:mail
```

If a backslash occurs in the middle of a line in a quoted string, the following escapes are recognized:

`\\`	single backslash
`\n`	newline
`\r`	carriage return
`\t`	tab
`\`<*octal digits*>	up to 3 octal digits specify one character
`\x`<*hex digits*>	up to 2 hexadecimal digits specify one character

If a backslash is followed by some other character, including a double-quote character, then that character replaces the pair.

7.9 Expanded strings

Some strings in the configuration file are subjected to *string expansion*, by which means various parts of the string may be changed according to the circumstances. The input syntax for such strings is as just described; the expansion process is described in chapter 9.

7.10 User and group names

User and group names are specified as strings, using the syntax described above, but the strings are interpreted specially. In the main section of the configuration file, a user or group name must either consist entirely of digits, or be a name that can be looked up using the **getpwnam()** or **getgrnam()** function, as appropriate.

When a user or group is specified as an option for a driver, it may alternatively be a string that gets expanded each time the user or group value is required. The presence of a **$** character in the string causes this action to happen. Each time the string is expanded, the result must either be a digit string, or a name that can be looked up using **getpwnam()** or **getgrnam()**, as appropriate.

7.11 String lists

Some configuration settings accept a colon-separated list of strings. In these cases the entire list is treated as a single string as far as the input syntax is concerned. The **trusted_users** setting in section 7.8 above is an example. If a colon is actually needed in an item in a string list, it can be entered as two colons. This is unfortunately necessary for all colons appearing in IPv6 addresses.

Leading and trailing white space on each item in a string list is ignored. This makes it possible to include items that start with a colon, and in particular, certain forms of IPv6 address. For example:

```
  local_interfaces = "127.0.0.1 : ::::1"
```

See **README.IPV6** for general information about IPv6 support in Exim.

7.12 Domain lists

Domain lists are colon-separated string lists containing a number of patterns that are to be matched against a mail domain. For example, the **local_domains** option is a domain list which must match all the domains that Exim is to treat as local.

Items in a domain list may be positive or negative. Negative items are indicated by a leading exclamation mark, which may be followed by optional white space. The list is scanned from left to right. If the domain matches a positive item, it is in the set of domains which the list defines; if it matches a negative item, it is not in the set. If the end of the list is reached without the domain having matched any of the patterns, it is accepted if the last item was a negative one, but not if it was a positive one. For example,

```
  relay_domains = !a.b.c : *.b.c
```

matches any domain ending in **.b.c** except for **a.b.c**. Domains that match neither **a.b.c** nor ***.b.c** are not accepted, because the last item in the list is positive. However, if the setting were

```
  relay_domains = !a.b.c
```

then all domains other than **a.b.c** would be accepted because the last item in the list is negative. In effect, a list that ends with a negative item behaves as if it had ': *' appended to it.

The following types of item may appear in domain lists:

- If an item in a domain list is a plain absolute file name (beginning with a slash character), then each line of the file is read and processed as if it were an independent item in the list, except that further plain file names are not allowed. This happens each time the list is searched. If a # character appears anywhere in a line of the file, it and all following characters are ignored. Blank lines are also ignored. Wild cards, negation, and regular expressions may be used in the lines of the file, just as in the main list. For example, if

```
local_domains = /etc/local-domains
```

then the file could contain lines like

```
^.*\d{3}\.mydomain\.com$
```

If a plain file name is preceded by an exclamation mark, the sense of any match within the file is inverted. For example, if

```
hold_domains = !/etc/nohold-domains
```

and the file contains the lines

```
!a.b.c
*.b.c
```

then **a.b.c** is in the set of domains defined by **hold_domains**, whereas any domain matching ***.b.c** is not.

- If a pattern consists of a single @ character, it matches the local host name, as set in the **primary_hostname** option. This makes it possible to use the same configuration file on several different hosts that differ only in their names.
- If a pattern starts with an asterisk, then the remaining characters of the pattern are compared with the terminating characters of the domain. The use of '*' in domain lists differs from its use in partial matching lookups. In a domain list, the character following the asterisk need not be a dot, whereas partial matching works only in terms of dot-separated components. For example, a domain list item such as ***key.ex** matches **donkey.ex** as well as **cipher.key.ex**.
- If a pattern starts with a circumflex character, then it is treated as a regular expression, and matched against the domain using a regular expression matching function. The circumflex is treated as part of the regular expression. References to descriptions of the syntax of regular expressions are given in chapter 8, but note that if a backslash is required in the regular expression, it must be given as two backslashes if the string is in quotes.

  There are some cases where a domain list is the result of string expansion, for example the **domains** option in routers and directors. In these cases you must escape any backslash and dollar characters in regular expressions, to prevent them from being interpreted by the string expander, and if the string is specified in quotes, the resulting backslashes must themselves also be escaped.
- If a pattern starts with the name of a single-key lookup type followed by a semicolon (for example, 'dbm;' or 'lsearch;') then the remainder of the pattern must be a file name in a suitable format for the lookup type. For example, for 'lsearch;' it must be an absolute path. The appropriate type of lookup is done on the file using the domain name as the key. The data from the lookup is available in some cases via the expansion variable **$domain_data**. Note that this is not an 'include' facility when the lookup type is 'lsearch'. The keys in the file are not interpreted specially, as they would be if they appeared as individual items in the domain list, or as lines in a file referenced without a search type.
- Any of the single-key lookup type names may be preceded by 'partial<*n*>-', where the <*n*> is optional, for example,

  ```
  partial-dbm;/partial/domains
  ```

  This causes partial matching logic to be invoked; a description of how this works is given in the next section.
- Also, any of the single-key lookup types may be followed by an asterisk. This causes a default lookup for a key consisting of a single asterisk to be done if the original lookup fails. This is not a useful feature when using a domain list to select particular domains (because any domain would match), but it might have value if the result of the lookup is being used via the **$domain_data** expansion variable.
- If the pattern starts with the name of a query-style lookup type followed by a semicolon (for example, 'nisplus;' or 'ldap;') then the remainder of the pattern must be an appropriate query for

the lookup type, as described in chapter 6. The data returned by a successful query is available in some cases via the expansion variable **$domain_data**.

The query is expanded before use, and the expansion substitution **$key** can be used to insert the domain that is being tested into the query. There are cases where a domain list is the result of an earlier string expansion, for example the **domains** option in routers and directors. In these cases you must use **\$key** to delay the substitution of the variable until the second expansion, and a double backslash is needed if the whole domain list is in quotes.

- If none of the above cases apply, a straight textual comparison is made between the pattern and the domain.

Here is an example which uses several different kinds of pattern:

```
local_domains = "@@:\
                 lib.unseen.edu:\
                 *.foundation.fict.book:\
                 ^[1-2]\\d{3}\\.fict\\.book$:\
                 dbm;/opt/data/penguin/book:\
                 nis;domains.byname:\
                 nisplus;[name=$key,status=local],domains.org_dir"
```

Note the necessary doubling of the backslashes in the regular expression. There are obvious processing trade-offs among the various matching modes. Using an asterisk is faster than a regular expression, and listing a few names explicitly probably is too. The use of a file or database lookup is expensive, but may be the only option if hundreds of names are required. Because the patterns are tested in order, it makes sense to put the most commonly matched patterns earlier in the string.

7.13 Partial matching in domain lists

When one of the single-key lookup types is preceded by ‘partial-’ then matching proceeds as follows: First the subject text is looked up verbatim; if that fails, ‘*.’ is added to the front of the subject and another lookup is tried. If that fails, domains are chopped off and replaced by ‘*.’ until there are fewer than two left. For example, if

```
*.neverwhere.tvs
```

is a key in single-key lookup file, then subdomains of **neverwhere.tvs** such as **market.neverwhere.tvs** and **downst.neverwhere.tvs** match it, as does **neverwhere.tvs** itself, provided there isn’t a separate entry for it in the file. A different minimum number of components can be imposed by supplying a number after ‘partial’, for example, ‘partial3-dbm’.

7.14 Address lists

An address list is a string list in which each item is a pattern to be matched against a mail address. As in the case of domain lists, the list is searched from left to right, any item may be preceded by an exclamation mark to negate it, and a plain file name may appear as an entire item, causing each line of the file to be read and treated as a separate pattern. Because local parts may legitimately contain # characters, a comment in the file is recognized only if # is followed by white space or the end of the line.

The following kinds of pattern may appear inline or as lines in an included file:

- If a pattern starts with ^ then a regular expression match is done against the complete address, using the entire pattern as the regular expression.
- Otherwise, if there is no @ in the pattern, it is first matched against the domain part of the subject address, the local part being ignored. This match is done exactly as for an entry in a domain list, so, for example, the item may begin with * or it may be a (partial) lookup (see section 7.12). If there is no match, and the pattern consists of a single lookup, then the entire subject address is looked up in the file, with partial matching disabled. This means that an item such as

```
sender_reject_recipients = partial-dbm;/black/list
```

can reference a single file whose keys are a mixture of complete domains, partial domains, and individual mail addresses. Note that this is not an 'include' facility when the lookup type is 'lsearch'. The keys in the file are not interpreted specially, as they would be if they appeared as individual items in the address list, or lines in a file given as a plain file name without a search type.

- If the pattern starts with '@@<lookup-item>' (for example, '@@lsearch;/some/file'), the address that is being checked is split into a local part and a domain. The domain is looked up in the file. If it is not found, there is no match. If it is found, the data that is looked up from the file is treated as a colon-separated list of local part patterns, each of which is matched against the subject local part in turn.

  The lookup may be a partial one, and/or one involving a search for a default keyed by '*'. The local part patterns that are looked up can be regular expressions or begin with '*', or even be further lookups. They may also be independently negated. For example, with

  ```
  sender_reject_recipients = @@dbm;/etc/reject-by-domain
  ```

  the data from which DBM file is built could contain lines like

  ```
  baddomain.com:  !postmaster : *
  ```

  If a local part that actually begins with an exclamation mark is required, it has to be specified using a regular expression. In **lsearch** files, an entry may be split over several lines by indenting the second and subsequent lines, but the separating colon must still be included at line breaks. White space surrounding the colons is ignored. For example:

  ```
  aol.com:  spammer1 : spammer2 : ^[0-9]+$ :
            spammer3 : spammer4
  ```

  As in all colon-separated lists in Exim, a colon can be included in an item by doubling.

  If the last item in the list starts with a right angle-bracket, then the remainder of the item is taken as a new key to look up in order to obtain a continuation list of local parts. The new key can be any sequence of characters. Thus one might have entries like

  ```
  aol.com: spammer1 : spammer 2 : >*
  xyz.com: spammer3 : >*
  *:       ^\d{8}$
  ```

  in a file that was searched with **@@dbm***, to specify a match for 8-digit local parts for all domains, in addition to the specific local parts listed for each domain. Of course, using this feature costs another lookup each time a chain is followed, but the effort needed to maintain the data is reduced. It is possible to construct loops using this facility, and in order to catch them, the chains may be no more than fifty items long.

- If none of the above cases apply, the local part of the subject address is compared with the local part of the pattern, which may start with an asterisk. If the local parts match, then the domains are compared in exactly the same way as entries in a domain list, except that a regular expression is not permitted for a domain only. However, file lookups are permitted. For example:

  ```
  sender_reject = "*@*.spamming.site:\
                   bozo@partial-lsearch;/list/of/dodgy/sites"
  ```

  The domain may be given as a single @ character, as in a domain list, standing for the local host name, leading to items of the form 'user@@'. If a local part that actually begins with an exclamation mark is required, it has to be specified using a regular expression, as otherwise the exclamation mark is treated as a sign of negation.

7.15 Case of letters in address lists

Domains in email addresses are always handled caselessly, but for local parts case may be significant on some systems (see **locally_caseless** for how Exim deals with this when processing local addresses). However, RFC 2505 (*Anti-Spam Recommendations for SMTP MTAs*) suggests that matching of addresses to blocking lists should be done in a case-independent manner. Since most address lists in Exim are used for this kind of control, Exim attempts to do this by default.

The domain portion of an address is always lowercased before matching it to an address list. The local part is lowercased by default, and any string comparisons that take place are done caselessly. This means that the data in the address list itself, in files included as plain file names, and in any file that is looked up using the '@@' mechanism, can be in any case. However, the keys in files that are looked up by a search type other than **lsearch** (which works caselessly) must be in lower case, because these lookups are not case-independent.

To allow for the possibility of caseful address list matching, if an item in the list is the string '+caseful' then the original case of the local part is restored for any comparisons that follow, and string comparisons are no longer case-independent. This does not affect the domain.

7.16 Host lists

Host lists are used to control what remote hosts are allowed to do (for example, use the local host as a relay). A host list consists of host name and IP address patterns which define a set of hosts that the list matches. Items in the list may be positive or negative. Negation is indicated by preceding an item with an exclamation mark. A plain absolute file name (beginning with a slash) can be used to include items from a file. Negation and included files operate exactly as for domain lists – see section 7.12 for examples.

The following types of pattern may appear in a host list:

- If the entire item is '*' it matches any host.
- If the item is in the form of an IP address, it is matched against the IP address of the subject host. The presence of a colon is taken as an indication that it is an IPv6 address (when IPv6 support is compiled into Exim); such colons have to be doubled, because colon is the item separator in the list.
- If the item is in the form of an IP address followed by a slash and a mask length (for example 131.111.0.0/16) then it is matched against the IP address of the subject host under the given mask, which specifies the number of address bits that must match, starting from the most significant end. Thus an entire network of hosts can be included (or excluded) by a single item.

  IPv4 addresses are given in the normal 'dotted-quad' notation. IPv6 addresses are given in colon-separated format, but the colons have to be doubled so as not to be taken as item separators. This example shows both kinds of address:

  ```
  receiver_unqualified_hosts = "123.123.0.0/16: \
                                 5f03::1200::836f::::/48"
  ```

  The doubling of colons in IPv6 addresses applies only when such addresses appear inline in a host list. It is not required when indirecting via a file. For example,

  ```
  receiver_unqualified_hosts = /opt/exim/unqualnets
  ```

  could make use of a file containing

  ```
  123.123.0.0/16
  5f03:1200:836f::/48
  ```

  to have exactly the same effect as the previous example.

  If an IPv4 host calls an IPv6 host, the incoming address actually appears in the IPv6 host as '::`ffff`:<*v4address*>'. When such an address is tested against a host list, it is converted into a traditional IPv4 address first.

- If the item is of the form

  `net`*<number>*`-`*<search-type>*`;`*<search-data>*

  for example:

  ```
  net24-dbm;/networks.db
  ```

  then the IP address of the subject host is masked using <*number*> as the mask length; a textual string is then constructed from the masked value, followed by the mask, and this is then used as the key for the lookup. For example, if the host' s IP address is 192.152.34.6 then the key that is looked up for the above example is '192.152.34.0/24'.

- If the entire item is '@' the primary host name is used as the the match item, and the following applies:

- If the item is a plain domain name, then Exim calls **gethostbyname()** to find its IP address(es). This typically causes a forward DNS lookup of the name. The result is compared with the IP address of the subject host.

The remaining items are wildcarded patterns for matching against the host name. If the host name is not already known, Exim calls **gethostbyaddr()** to obtain it from the IP address. This typically causes a reverse DNS lookup to occur. If the lookup fails, Exim takes a hard line by default and access is not permitted. If the list is an 'accept' list, Exim behaves as if the current host is not in the set defined by the list, whereas if it is a 'reject' list, it behaves as if it is. For example,

```
host_reject = +allow_unknown:*.enemy.ex
```

rejects connections from any host whose name matches **∗.enemy.ex**, but only if it can find a host name from the incoming IP address.

To change this behaviour, the special item '+allow_unknown' may appear in the list (at top level – it is not recognized in an indirected file); if any subsequent items require a host name, and the reverse DNS lookup fails, Exim permits the access, that is, its behaviour is the opposite to the default.

As a result of aliasing, hosts may have more than one name. When processing any of the following items, all the host's names are checked.

- If the item starts with '∗' then the remainder of the item must match the end of the host name. For example, **∗.b.c** matches all hosts whose names end in **.b.c**. This special simple form is provided because this is a very common requirement. Other kinds of wildcarding require the use of a regular expression.

- If the item starts with '^' then it is taken to be a regular expression which is matched against the host name. For example,

  ```
  ^(a|b)\.c\.d$
  ```

  matches either of the two hosts **a.c.d** or **b.c.d**. If the option string in which this occurs is given in quotes, then the backslash characters must be doubled, because they are significant in quoted strings. The following two settings are exactly equivalent:

  ```
  host_accept = ^(a|b)\.c\.d$
  host_accept = "^(a|b)\\.c\\.d$"
  ```

- If the item is of the form

  <search-type>`;`*<filename or query>*

  for example

  ```
  dbm;/host/accept/list
  ```

  then the host name is looked up using the search type and file name or query (as appropriate). If the lookup succeeds, the host matches the item. The actual data that is looked up is not used.

7.17 Use of RFC 1413 identification in host lists

Any item in a host list (other than a plain file name or '+allow_unknown') can optionally be preceded by

> <*ident*>`@`
> or
> `!`<*ident*>`@`

where <*ident*> is an RFC 1413 identification string. For example,

```
host_reject = !exim@my.mail.gate:111.111.111.111:!root@public.host
```

If an <*ident*> string is present, it must match the RFC 1413 identification sent by the remote host, unless it is preceded by an exclamation mark, in which case it must *not* match. The remainder of the item, following the @, may be either positive or negative.

8. Regular expressions

Exim uses the PCRE regular expression library; this provides regular expression matching that is compatible with Perl 5. The syntax and semantics of these regular expressions is discussed in many Perl reference books, and also in Jeffrey Friedl's *Mastering Regular Expressions* (O'Reilly, ISBN 1-56592-257-3).

The PCRE distribution files, which are included in the directory **src/pcre** in the Exim distribution, contain a man page for PCRE which describes exactly what it supports, so no further description is included here. The PCRE functions are called from Exim using the default option settings, except that the PCRE_CASELESS option is set when the matching is required to be independent of the case of letters.

8.1 Testing regular expressions

A program called **pcretest** forms part of the PCRE distribution and is built with PCRE during the process of building Exim. It is primarily intended for testing PCRE itself, but it can also be used for experimenting with regular expressions. The binary can be found in the **pcre** sub-directory of the Exim build directory. There is documentation of various options in **src/pcre/README**, but for simple testing, none are needed. This is the output of a sample run of **pcretest**:

```
  re> /^([^@]+)@.+\.(ac|edu)\.(?!kr)[a-z]{2}$/
  data> x@y.ac.uk
 0: x@y.ac.uk
 1: x
 2: ac
  data> x@y.ac.kr
No match
  data> x@y.edu.com
No match
  data> x@y.edu.co
 0: x@y.edu.co
 1: x
 2: edu
```

Input typed by the user is shown in bold face. After the 're>' prompt, a regular expression enclosed in delimiters is expected. If this compiles without error, 'data>' prompts are given for strings against which the expression is matched. An empty data line causes a new regular expression to be read. If the match is successful, the captured substring values (that is, what would be in the variables **$0**, **$1**, **$2**, etc.) are shown. The above example tests for an email address whose domain ends with either 'ac' or 'edu' followed by a two-character top-level domain that is not 'kr'. The local part is captured in **$1** and the 'ac' or 'edu' in **$2**.

[46] *regular expressions (8)*

9. String expansions

A number of configuration strings are expanded before use. Some of them are expanded every time they are used; others are expanded only once.

Expanded strings are copied verbatim except when a dollar or backslash character is encountered. A dollar specifies the start of a portion of the string which is interpreted and replaced as described below.

An uninterpreted dollar can be included in the string by putting a backslash in front of it – if the string appears in quotes, two backslashes are required because the quotes themselves cause interpretation of backslashes when the string is read in. A backslash can be used to prevent any special character being treated specially in an expansion, including itself.

A backslash followed by one of the letters 'n', 'r', or 't' is recognized as an escape sequence for the character newline, carriage return, or tab, respectively. A backslash followed by up to three octal digits is recognized as an octal encoding for a single character, while a backslash followed by 'x' and up to two hexadecimal digits is a hexadecimal encoding. A backslash followed by any other character causes that character to be added to the output string uninterpreted. These escape sequences are also recognized in quoted strings; their interpretation in expansions is useful for unquoted strings and other cases such as looked-up strings that are then expanded.

9.1 Testing string expansions

A program to test string expansions can be compiled by obeying the command

```
make test_expand
```

once Exim has been successfully compiled. This makes a binary called **test_expand** in the build directory. When run, it reads lines from the standard input, runs them through the string expansion code, and writes the results to the standard output. Since no message is being processed, variables such as **$local_part** have no value, but the program can be used for checking out file and database lookups, and the use of expansion operators such as **substr** and **hash**.

9.2 Expansion items

The following items are recognized in expanded strings. White space may be used between sub-items that are keywords or substrings enclosed in braces inside an outer set of braces, to improve readability.

$**<*variable name*> or **${<*variable name*>**}**

Substitute the contents of the named variable; the latter form can be used to separate the name from subsequent alphanumeric characters. This form (using curly brackets) is available only for variables; it does *not* apply to message headers. The names of the variables are given in section 9.5 below. If the name of a non-existent variable is given, the expansion fails.

$header_<*header name*>**:** or **$h_**<*header name*>**:**

Substitute the contents of the named message header, for example

```
$header_reply-to:
```

This particular expansion is intended mainly for use in filter files. The header names follow the syntax of RFC 822, which states that they may contain any printing characters except space and colon. Consequently, curly brackets *do not* terminate header names, and should not be used to enclose them as if they were variables. Attempting to do so causes a syntax error.

Upper-case and lower-case letters are synonymous in header names. If the following character is white space, the terminating colon may be omitted. The white space is included in the expanded string. If the message does not contain the given header, the expansion item is replaced by an empty string. (See the **def** condition in section 9.4 for a means of testing for the existence of a header.) If there is more than one header with the same name, they are all concatenated to form the substitution string, with a newline character between each of them. However, if the length of this string exceeds 64K, any further headers of the same name are ignored.

${<*op*>**:**<*string*>**}**

The string is first itself expanded, and then the operation specified by <*op*> is applied to it. A list of operators is given in section 9.3 below. The string starts with the first character after the colon, which may be leading white space.

${if <*condition*> **{**<*string1*>**}{**<*string2*>**}}**

If <*condition*> is true, <*string1*> is expanded and replaces the whole item; otherwise <*string2*> is used. The second string need not be present; if it is not and the condition is not true, the item is replaced with nothing. Alternatively, the word 'fail' may be present instead of the second string (without any curly brackets). In this case, the expansion fails if the condition is not true. The available conditions are described in section 9.4 below.

${lookup{<*key*>**}** <*search type*> **{**<*file*>**}** **{**<*string1*>**}** **{**<*string2*>**}}**

${lookup <*search type*> **{**<*query*>**}** **{**<*string1*>**}** **{**<*string2*>**}}**

These items specify data lookups in files and databases, as discussed in chapter 6. The first form is used for single-key lookups, and the second is used for query-style lookups. The <*key*>, <*file*>, and <*query*> strings are expanded before use.

If there is any white space in a lookup item which is part of a filter command, a rewrite rule, a routing rule for the **domainlist** router, or any other place where white space is significant, the lookup item must be enclosed in double quotes.

If the lookup succeeds, then <*string1*> is expanded and replaces the entire item. During its expansion, a variable called **$value** is available, containing the data returned by the file lookup. If the lookup fails, <*string2*> is expanded and replaces the entire item. It may be omitted, in which case the replacement is null.

For single-key lookups, the string 'partial-' is permitted to precede the search type in order to do partial matching, and * or *@ may follow a search type to request default lookups if the key does not match (see sections 6.1 and 6.5).

If a partial search is used, the variables **$1** and **$2** contain the wild and non-wild parts of the key during the expansion of the replacement text. They return to their previous values at the end of the lookup item.

Instead of {<*string2*>} the word 'fail' can appear, and in this case, if the lookup fails, the entire string expansion fails in a way that can be detected by the code in Exim which requested the expansion. The consequences of this depend on the circumstances. In some cases it is no different from any other expansion failure, but in others a different action may be taken. See for example the **new_address** option of the **smartuser** director.

This example looks up the postmaster alias in the conventional alias file.

```
"${lookup {postmaster} lsearch {/etc/aliases} {$value}}"
```

This example uses NIS+ to look up the full name of the user corresponding to the local part of an address, failing the expansion if it is not found.

```
"${lookup nisplus {[name=$local_part],passwd.org_dir:gcos} \
  {$value}fail}"
```

${lookup{<*key:subkey*>**}** <*search type*> **{**<*file*>**}** **{**<*string1*>**}** **{**<*string2*>**}}**

This searches for <*key*> in the file as described above for single-key lookups; if it succeeds, it extracts from the data a subfield which is identified by the <*subkey*>. The data related to the main key must be of the form:

<*subkey1*> = <*value1*>   <*subkey2*> = <*value2*> ...

where the equals signs are optional. If any of the values contain white space, they must be enclosed in double quotes, and any values that are enclosed in double quotes are subject to escape processing as described in section 7.8. For example, if a line in a linearly searched file contains

```
alice: uid=1984 gid=2001
```

then expanding the string

`${lookup{alice:uid}lsearch{`<*file name*>`}{$value}}`

yields the string ‘1984’. If the subkey is not found in <*string1*>, then <*string2*>, if present, is expanded and replaces the entire item. Otherwise the replacement is null.

${extract{<*key*>**} {**<*string*>**}}**

The key and the string are first expanded. Then the subfield identified by the key is extracted from the string, exactly as just described for **lookup** items with subkeys. If the key is not found in the string, the item is replaced by nothing.

${extract{<*number*>**} {**<*separators*>**} {**<*string*>**}}**

This is distinguished from the above form of **extract** by having three rather than two arguments. It extracts from the string the field whose number is given as the first argument. The first field is numbered one. If the number is negative or greater than the number of fields in the string, the result is empty; if it is zero the entire string is returned. The fields in the string are separated by any one of the characters in the separator string. For example:

```
${extract{3}{:}{exim:x:42:99:& Mailer::/bin/bash}}
```

yields ‘42’. Two successive separators mean that the field between them is empty (for example, the sixth field above). If the first argument is not numeric, the expansion fails.

${perl{<*subroutine*>**}{**<*arg*>**}{**<*arg*>**}...}**

This item is available only if Exim has been built to include an embedded Perl interpreter. The subroutine name and the arguments are first separately expanded, and then the Perl subroutine is called with those arguments. No arguments need be given; the maximum number permitted is eight.

The return value of the subroutine is inserted into the expanded string, unless the return value is **undef**. In that case, the expansion fails in the same way as an explicit ‘fail’ on a lookup item. If the subroutine exits by calling Perl’s **die** function, the expansion fails with the error message that was passed to **die**.

More details of the embedded Perl facility are given in chapter 10.

9.3 Expansion operators

The following operations can be performed on portions of an expanded string:

${domain:<*string*>**}**

The string is interpreted as an RFC 822 address and the domain is extracted from it. If the string does not parse successfully, the result is empty.

${escape:<*string*>**}**

If the string contains any non-printing characters, they are converted to escape sequences starting with a backslash. Whether characters with the most significant bit set (so-called ‘8-bit characters’) count as printing or not is controlled by the **print_topbitchars** option.

${expand:<*string*>**}**

The **expand** operator causes a string to be expanded for a second time. For example,

```
${expand:${lookup{$domain}dbm{/some/file}{$value}}}
```

first looks up a string in a file while expanding the operand for **expand**, and then re-expands what it has found.

${hash_<*n*>**_**<*m*>**:**<*string*>**}**

The two items <*n*> and <*m*> are numbers. If <*n*> is greater than or equal to the length of the string, the operator returns the string. Otherwise it computes a new string of length <*n*> by applying a hashing function to the string. The new string consists of characters taken from the first <*m*> characters of the string

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQWRSTUVWXYZ0123456789
```

and if <*m*> is not present the value 26 is used, so that only lower case letters appear. These examples:

```
${hash_3:monty}
${hash_5:monty}
${hash_4_62:monty python}
```

yield

```
jmg
monty
fbWx
```

respectively. The abbreviation **h** can be used instead of **hash**.

${lc:<*string*>**}**

This forces the letters in the string into lower-case, for example:

```
${lc:$local_part}
```

${uc:<*string*>**}**

This forces the letters in the string into upper-case.

${length_<*number*>**:**<*string*>**}**

The **length** operator can be used to extract the initial portion of a string. It is followed by an underscore and the number of characters required. For example

```
${length_50:$message_body}
```

The result of this operator is either the first <*number*> characters or the whole string, whichever is the shorter. The abbreviation **l** can be used instead of **length**.

${local_part:<*string*>**}**

The string is interpreted as an RFC 822 address and the local part is extracted from it. If the string does not parse successfully, the result is empty.

${quote:<*string*>**}**

The **quote** operator puts its argument into double quotes if it contains anything other than letters, digits, underscores, full stops (periods), and hyphens. Any occurrences of double quotes and backslashes are escaped with a backslash. For example,

```
${quote:ab*cd}
```

becomes

```
"ab*cd"
```

The place where this is useful is when the argument is a substitution from a variable or a message header.

${rxquote:<*string*>**}**

The **rxquote** operator inserts a backslash before any non-alphanumeric characters in its argument. This is useful when substituting the values of variables or headers inside regular expressions.

${substr_<*start*>**_**<*length*>**:**<*string*>**}**

The **substr** operator can be used to extract more general substrings than **length**. It is followed by an underscore and the starting offset, then a second underscore and the length required. For example

```
${substr_3_2:$local_part}
```

If the starting offset is greater than the string length the result is the null string; if the length plus starting offset is greater than the string length, the result is the right-hand part of the string, starting from the given offset. The first character in the string has offset zero. The abbreviation **s** can be used instead of **substr**.

The **substr** expansion operator can take negative offset values to count from the righthand end of its operand. The last character is offset -1, the second-last is offset -2, and so on. Thus, for example,

```
${substr_-5_2:1234567}
```

yields '34'. If the absolute value of a negative offset is greater than the length of the string, the substring starts at the beginning of the string, and the length is reduced by the amount of overshoot. Thus, for example,

```
${substr_-5_2:12}
```

yields an empty string, but

```
${substr_-3_2:12}
```

yields '1'.

If the second number is omitted from **substr**, the remainder of the string is taken if the offset was positive. If it was negative, all characters in the string preceding the offset point are taken. For example, an offset of -1 and no length yields all but the last character of the string.

9.4 Expansion conditions

The following conditions are available for testing while expanding strings:

!<*condition*>

This negates the result of the condition.

<*symbolic operator*> **{**<*string1*>**}{**<*string2*>**}**

There are a number of symbolic operators for doing numeric comparisons. They are:

=	equal
==	equal
>	greater
>=	greater or equal
<	less
<=	less or equal

Note that the general negation operator provides for inequality testing. The two strings must take the form of optionally signed decimal integers, optionally followed by one of the letters 'K' or 'M' (in either case), signifying multiplication by 1024 or 1024∗1024, respectively.

def:<*variable name*>

The **def** condition must be followed by the name of one of the expansion variables defined in section 5. The condition is true if the named expansion variable does not contain the empty string, for example

```
${if def:sender_ident {from $sender_ident}}
```

Note that the variable name is given without a leading **$** character. If the variable does not exist, the expansion fails.

def:header_<*header name*>**:** **or** **def:h_**<*header name*>**:**

This condition is true if a message is being processed and the named header exists in the message. For example,

```
${if def:header_reply-to:{$h_reply-to:}{$h_from:}}
```

Note that no **$** appears before **header_** or **h_** in the condition, and that header names must be terminated by colons if white space does not follow.

exists {<*file name*>**}**

The substring is first expanded and then interpreted as an absolute path. The condition is true if the named file (or directory) exists. The existence test is done by calling the **stat()** function.

eq {<*string1*>**}{**<*string2*>**}**

The two substrings are first expanded. The condition is true if the two resulting strings are identical, including the case of letters.

match {<*string1*>**}{**<*string2*>**}**

The two substrings are first expanded. The second is then treated as a regular expression and applied to the first. Because of the pre-expansion, if the regular expression contains dollar or backslash characters, they must be escaped with backslashes. Care must also be taken if the regular expression contains braces (curly brackets). A closing brace must be escaped so that it is not taken as a premature termination of <*string2*>. It does no harm to escape opening braces, but this is not strictly necessary. If the whole expansion string is in double quotes, further escaping of backslashes is also required.

The condition is true if the regular expression match succeeds. At the start of an **if** expansion the values of the numeric variable substitutions **$1** etc. are remembered. Obeying a **match** condition that succeeds causes them to be reset to the substrings of that condition and they will have these values during the expansion of the success string. At the end of the **if** expansion, the previous values are restored. After testing a combination of conditions using **or**, the subsequent values of the numeric variables are those of the condition that succeeded.

or {{<*cond1*>**}{**<*cond2*>**}...}**

The sub-conditions are evaluated from left to right. The condition is true if any one of the sub-conditions is true. When a true sub-condition is found, the following ones are parsed but not evaluated. Thus if there are several 'match' sub-conditions the values of the numeric variables are taken from the first one that succeeds.

and {{<*cond1*>**}{**<*cond2*>**}...}**

The sub-conditions are evaluated from left to right. The condition is true if all of the sub-conditions are true. When a false sub-condition is found, the following ones are parsed but not evaluated.

9.5 Expansion variables

The variable substitutions that are available for use in expansion strings are:

$0, **$1**, etc: When a **matches** expansion condition succeeds, these variables contain the captured substrings identified by the regular expression during subsequent processing of the success string of the containing **if** expansion item. They may also be set externally by some other matching process which precedes the expansion of the string. For example, the commands available in Exim filter files include an **if** command with its own regular expression matching condition.

$address_file: When, as a result of aliasing or forwarding, a message is directed to a specific file, this variable holds the name of the file when the transport is running. For example, using the default configuration, if user **r2d2** has a **.forward** file containing

```
/home/r2d2/savemail
```

then when the **address_file** transport is running, **$address_file** contains '/home/r2d2/savemail'. At other times, the variable is empty.

$address_pipe: When, as a result of aliasing or forwarding, a message is directed to a pipe, this variable holds the pipe command when the transport is running.

$caller_gid: The group id under which the process that called Exim was running. This is not the same as the group id of the originator of a message (see **$originator_gid**). If Exim re-execs itself, this variable in the new incarnation normally contains the Exim gid.

$caller_uid: The user id under which the process that called Exim was running. This is not the same as the user id of the originator of a message (see **$originator_uid**). If Exim re-execs itself, this variable in the new incarnation normally contains the Exim uid.

$compile_date: The date on which the Exim binary was compiled.

$compile_number: The building process for Exim keeps a count of the number of times it has been compiled. This serves to distinguish different compilations of the same version of the program.

$domain: When an address is being directed, routed, or delivered on its own, this variable contains the domain. In particular, it is set during user filtering, but not during system filtering, since a message may have many recipients and the system filter is called just once.

For remote addresses, the domain that is being routed can change as routing proceeds, as a result of router actions (see, for example, the **domainlist** router). However, the value of **$domain** remains as the original domain. The current routing domain can often be accessed by ot her means.

When a remote or local delivery is taking place, if all the addresses that are being handled simultaneously contain the same domain, it is placed in **$domain**. Otherwise this variable is empty. Transports should be restricted to handling only one domain at once if its value is required at transport time – this is the default for local transports. For further details of the environment in which local transports are run, see chapter 13.

Because configured address rewriting happens at the time a message is received, **$domain** normally contains the value after rewriting. However, when a rewrite item is actually being processed (see chapter 32) **$domain** contains the domain portion of the address that is being rewritten; it can be used in the expansion of the replacement address, for example, to rewrite domains by file lookup.

When the **smtp_etrn_command** option is being expanded, **$domain** contains the complete argument of the ETRN command (see section 42.6).

$domain_data: When a director or a router has a setting of the **domains** generic option, and that involves a file lookup, the data associated with the key in the file is available during the running of the director or router as **$domain_data**. In all other situations, this variable expands to nothing.

$errmsg_recipient: This is set to the recipient address of an error message while Exim is creating it. It is useful if a customized error message text file is in use (see chapter 33).

$home: A home directory may be set during a local delivery, either by the transport or by the director that handled the address. When this is the case, **$home** contains its value and may be used in any expanded options for the transport. The **forwardfile** director also makes use of **$home**. Full details are given in chapter 22. When interpreting a user's filter file, Exim is normally configured so that

$home contains the user's home directory. When running a filter test via the **-bf** option, **$home** is set to the value of the environment variable HOME.

$host: When a local transport is run as a result of routing a remote address, this variable is available to access the host name that the router defined. A router may set up many hosts; in this case **$host** refers to the first one. It is expected that this usage will be mainly via the domainlist router, setting up a single host for batched SMTP output, for example.

When used in a transport filter (see chapter 14) **$host** refers to the host involved in the current connection.

$host_address: This variable is available only for use in transport filters (see chapter 14).

$interface_address: For a message received over a TCP/IP connection, this variable contains the address of the IP interface that was used. See also the **-oMi** command line option.

$key: When a domain list is being searched, this variable contains the value of the key, so that it can be inserted into strings for query-style lookups. See chapter 6 for details. In other circumstances this variable is empty.

$local_part: When an address is being directed, routed, or delivered on its own, this variable contains the local part. If a local part prefix or suffix has been recognized, it is not included in the value. When a number of addresses are being delivered in a batch by a local or remote transport, **$local_part** is not set.

When a message is being delivered to a pipe, file, or autoreply transport as a result of aliasing or forwarding, **$local_part** is set to the local part of the parent address.

If an address is source-routed, that is, of the form

```
@a:c@d
```

then when its transport is running **$local_part** is set to 'c@d' and **$domain** is set to 'a'.

Because configured address rewriting happens at the time a message is received, **$local_part** normally contains the value after rewriting. However, when a rewrite item is actually being processed (see chapter 32) **$local_part** contains the local part of the address that is being rewritten; it can be used in the expansion of the replacement address, for example, to rewrite local parts by file lookup.

$local_part_data: When a director or a router has a setting of the **local_parts** generic option, and that involves a file lookup, the data associated with the key in the file is available during the running of the director or router as **$local_part_data**. In all other situations, this variable expands to nothing.

$local_part_prefix: When an address is being directed or delivered locally, and a specific prefix for the local part was recognized, it is available in this variable. Otherwise it is empty.

$local_part_suffix: When an address is being directed or delivered locally, and a specific suffix for the local part was recognized, it is available in this variable. Otherwise it is empty.

$localhost_number: This contains the expanded value of the **localhost_number** option. The expansion happens after the main options have been read.

$message_body: This variable contains the initial portion of a message's body while it is being delivered, and is intended mainly for use in filter files. The maximum number of characters of the body that are used is set by the **message_body_visible** configuration option; the default is 500. Newlines are converted into spaces to make it easier to search for phrases that might be split over a line break.

$message_body_end: This variable contains the final portion of a message's body while it is being delivered. The format and maximum size are as for **$message_body**.

$message_body_size: When a message is being received or delivered, this variable contains the size of the body in bytes. The count starts from the character after the blank line that separates the body from the header. Newlines are included in the count. See also **$message_size**.

$message_headers: This variable contains a concatenation of all the header lines when a message is being processed. They are separated by newline characters.

$message_id: When a message is being received or delivered, this variable contains the unique message id which is used by Exim to identify the message.

$message_precedence: When a message is being delivered, the value of any **Precedence:** header is made available in this variable. If there is no such header, the value is the null string.

$message_size: When a message is being received or delivered, this variable contains its size in bytes. The size includes those headers that were received with the message, but not those (such as **Envelope-to:**) that are added to individual deliveries. See also **$message_body_size**.

$n0 – **$n9**: These variables are counters that can be incremented by means of the **add** command in filter files.

$original_domain: When a top-level address is being processed for delivery, this contains the same value as **$domain**. However, if a 'child' address (for example, generated by an alias, forward, or filter file) is being processed, this variable contains the domain of the original address. This differs from **$parent_domain** when there is more than one level of aliasing or forwarding. When more than one address is being delivered in a batch by a local or remote transport, **$original_domain** is not set.

Address rewriting happens as a message is received. Once it has happened, the previous form of the address is no longer accessible. It is the rewritten top-level address whose domain appears in this variable.

$original_local_part: When a top-level address is being processed for delivery, this contains the same value as **$local_part**. However, if a 'child' address (for example, generated by an alias, forward, or filter file) is being processed, this variable contains the local part of the original address. This differs from **$parent_local_part** when there is more than one level of aliasing or forwarding. When more than one address is being delivered in a batch by a local or remote transport, **$original_local_part** is not set.

Address rewriting happens as a message is received. Once it has happened, the previous form of the address is no longer accessible. It is the rewritten top-level address whose local part appears in this variable.

$originator_gid: The value of **$caller_gid** that was set when the message was received. For messages received via the command line, this is the gid of the sending user. For messages received by SMTP over TCP/IP, this is normally the gid of the Exim user.

$originator_uid: The value of **$caller_uid** that was set when the message was received. For messages received via the command line, this is the uid of the sending user. For messages received by SMTP over TCP/IP, this is normally the uid of the Exim user.

$parent_domain: This variable is empty, except when a 'child' address (generated by aliasing or forwarding, for example) is being processed, in which case it contains the domain of the immediately preceding parent address.

$parent_local_part: This variable is empty, except when a 'child' address (generated by aliasing or forwarding, for example) is being processed, in which case it contains the local part of the immediately preceding parent address.

$pipe_addresses: This is not an expansion variable, but is mentioned here because the string '$pipe_addresses' is handled specially in the command specification for the **pipe** transport and in transport filters. It cannot be used in general expansion strings, and provokes an 'unknown variable' error if encountered.

$primary_hostname: The value set in the configuration file, or read by the **uname()** function.

$prohibition_reason: This variable is set only during the expansion of prohibition messages. See section 40.6 for details.

$qualify_domain: The value set for this option in the configuration file.

$qualify_recipient: The value set for this option in the configuration file, or if not set, the value of **$qualify_domain**.

$received_for: If there is only a single recipient address in an incoming message, then when the **Received:** header line is being built, this variable contains that address. Otherwise it is empty.

$received_protocol: When a message is being processed, this variable contains the name of the protocol by which it was received.

$recipients: This variable contains a list of envelope recipients for a message, but is recognized only in the system filter file, to prevent exposure of Bcc recipients to ordinary users. A comma and a space separate the addresses in the replacement text.

$recipients_count: When a message is being processed, this variable contains the number of envelope recipients that came with the message. Duplicates are not excluded from the count.

$reply_address: When a message is being processed, this variable contains the contents of the **Reply-to:** header if one exists, or otherwise the contents of the **From:** header.

$return_path: When a message is being delivered, this variable contains the return path – the sender field that is sent as part of the envelope. In many cases, this has the same value as **$sender_address**, but if, for example, an incoming message to a mailing list has been expanded by a director which specifies a specific address for delivery error messages, then **$return_path** contains the new error address, while **$sender_address** contains the original sender address that was received with the message.

$return_size_limit: This contains the value set in the **return_size_limit** option, rounded up to a multiple of 1000. It is useful when a customized error message text file is in use (see chapter 33).

$route_option: A router may set up an arbitrary string to be passed to a transport via this variable. Currently, only the **queryprogram** router has the ability to do so.

$self_hostname: The generic router option **self** can be set to the values 'local' or 'fail_soft' (amongst others). These cause the address to be passed over to the directors, as if its domain were a local domain, or to be passed on to the next router, respectively. While subsequently directing or routing (and doing any deliveries) **$self_hostname** is set to the name of the local host that the router encountered. In other circumstances its contents are null.

$sender_address: When a message is being processed, this variable contains the sender's address that was received in the message's envelope. For delivery failure reports, the value of this variable is the empty string.

$sender_address_domain: The domain portion of **$sender_address**.

$sender_address_local_part: The local part portion of **$sender_address**.

$sender_fullhost: When a message has been received from a remote host, this variable contains the host name and IP address in a single string, which always ends with the IP address in square brackets. The format of the rest of the string depends on whether the host issued a HELO or EHLO SMTP command, and whether the host name was verified by looking up its IP address. (Looking up the IP address can be forced by the **host_lookup** option, independent of verification.) A plain host name at the start of the string is a verified host name; if this is not present, verification either failed or was not requested. A host name in parentheses is the argument of a HELO or EHLO command. This is omitted if it is identical to the verified host name or to the host's IP address in square brackets.

$sender_helo_name: When a message has been received from a remote host that has issued a HELO or EHLO command, the first item in the argument of that command is placed in this variable. It is also set if HELO or EHLO is used when a message is received using SMTP locally via the **-bs** or **-bS** options.

$sender_host_address: When a message has been received from a remote host, this variable contains the host's IP address.

$sender_host_name: When a message has been received from a remote host, this variable contains the host's name as verified by looking up its IP address. If verification failed, or was not requested, this variable contains the empty string.

$sender_ident: When a message has been received from a remote host, this variable contains the identification received in response to an RFC 1413 request. When a message has been received locally, this variable contains the login name of the user that called Exim.

$sender_rcvhost: This is provided specifically for use in **Received:** headers. It starts with either the verified host name (as obtained from a reverse DNS lookup) or, if there is no verified host name, the IP address in square brackets. After that there may be text in parentheses. When the first item is a verified host name, the first thing in the parentheses is the IP address in square brackets. There may also be items of the form 'helo=*xxxx*' if HELO or EHLO was used and its argument was not identical to the real host name or IP address, and 'ident=*xxxx*' if an RFC 1413 ident string is available. If all three items are present in the parentheses, a newline and tab are inserted into the string, to improve the formatting of the **Received:** header.

$sn0 – **$sn9**: These variables are copies of the values of the **$n0** – **$n9** accumulators that were current at the end of the system filter file. This allows a system filter file to set values that can be tested in users' filter files. For example, a system filter could set a value indicating how likely it is that a message is junk mail.

$spool_directory: The name of Exim's spool directory.

$thisaddress: This variable is set only during the processing of the **foranyaddress** command in a filter file. Its use is explained in the description of that command.

$tod_bsdinbox: The time of day and date, in the format required for BSD-style mailbox files, for example: Thu Oct 17 17:14:09 1995.

$tod_full: A full version of the time and date, for example: Wed, 16 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, for example: 1995-10-12 15:32:29.

$value: This variable contains the result of an expansion lookup operation, as described above. Also, if a **domainlist** router has a lookup pattern in a route item, **$value** contains the data that was looked up during the expansion of the host list. If **$value** is used in other circumstances, its contents are null.

$version_number: The version number of Exim.

$warnmsg_delay: This variable is set only during the creation of a message warning about a delivery delay. Details of its use are explained in section 33.2.

$warnmsg_recipients: This variable is set only during the creation of a message warning about a delivery delay. Details of its use are explained in section 33.2.

9.6 Expansion string examples

Typical settings for defining a local mailbox to the **appendfile** transport are

```
  file = /var/spool/mail/${local_part}
  file = ${home}/inbox
```

As a more complicated example, the default setting for the **Received:** header is as follows:

```
received_header_text = "Received: \
    ${if def:sender_rcvhost {from ${sender_rcvhost}\n\t}\
    {${if def:sender_ident {from ${sender_ident} }}\
    ${if def:sender_helo_name {(helo=${sender_helo_name})\n\t}}}}\
    by ${primary_hostname} \
    ${if def:received_protocol {with ${received_protocol}}} \
    (Exim ${version_number} #${compile_number})\n\t\
    id ${message_id}
    ${if def:received_for {\n\tfor $received_for}}"
```

10. Embedded Perl

Exim can be built to include an embedded Perl interpreter. When this is done, Perl subroutines can be called as part of the string expansion process. To make use of the Perl support, you need version 5.004 or later of Perl installed on your system. To include the embedded interpreter in the Exim binary, include the line

```
  EXIM_PERL = perl.o
```

in your **Local/Makefile** and then build Exim in the normal way.

Access to Perl subroutines is via a global configuration option called **perl_startup** and an expansion string operator **${perl ...}**. If there is no **perl_startup** option in the Exim configuration file then no Perl interpreter is started and there is almost no overhead for Exim (since none of the Perl library will be paged in unless used). If there is a **perl_startup** option then the associated value is taken to be Perl code which is executed in a newly created Perl interpreter.

The value of **perl_startup** is not expanded in the Exim sense, so you do not need backslashes before any characters to escape special meanings. The option should usually be something like

```
  perl_startup = do '/etc/exim.pl'
```

where **/etc/exim.pl** is Perl code which defines any subroutines you want to use from Exim. Exim can be configured either to start up a Perl interpreter as soon as it is entered, or to wait until the first time it is needed. Starting the interpreter at the beginning ensures that it is done while Exim still has its setuid privilege, but can impose an unnecessary overhead if Perl is not in fact used in a particular run. By default, the interpreter is started only when it is needed, but this can be changed in two ways:

- Setting **perl_at_start** (a boolean option) in the configuration requests a startup when Exim is entered.
- The command line option **-ps** also requests a startup when Exim is entered, overriding the setting of **perl_at_start**.

There is also a command line option **-pd** (for delay) which suppresses the initial startup, even if **perl_at_start** is set.

When the configuration file includes a **perl_startup** option you can make use of the string expansion item to call the Perl subroutines that are defined by the **perl_startup** code. The operator is used in any of the following forms:

```
  ${perl{foo}}
  ${perl{foo}{argument}}
  ${perl{foo}{argument1}{argument2} ... }
```

which calls the subroutine **foo** with the given arguments. A maximum of eight arguments may be passed. Passing more than this results in an expansion failure with an error message of the form

```
  Too many arguments passed to Perl subroutine "foo" (max is 8)
```

The return value of the subroutine is inserted into the expanded string, unless the return value is **undef**. In that case, the expansion fails in the same way as an explicit 'fail' on an **${if ...}** or **${lookup...}** item. If the subroutine aborts by obeying Perl's **die** function, then the expansion fails with the error message that was passed to **die**.

Within any Perl code called from Exim, the function **Exim::expand_string** is available to call back into Exim's string expansion function. For example, the Perl code

```
  my $lp = Exim::expand_string('$local_part');
```

makes the current Exim **$local_part** available in the Perl variable **$lp**. Note those are single quotes and not double quotes to protect against **$local_part** being interpolated as a Perl variable.

If the string expansion is forced to fail by a ‘fail’ item, the result of **Exim::expand_string** is **undef**. If there is a syntax error in the expansion string, the Perl call from the original expansion string fails with an appropriate error message, in the same way as if **die** were used.

11. Main configuration

The first part of the runtime configuration file contains the main configuration settings. Each setting occupies one line of the file, except that string values can be continued onto multiple lines as described in section 7.8. All macro definitions must be in this part of the file – they differ from options settings by starting with an upper-case letter (see section 7.2). The available options are as follows:

accept_8bitmime

Type:	*boolean*
Default:	*false*

This option causes Exim to send 8BITMIME in its response to an SMTP EHLO command, and to accept the BODY= parameter on MAIL commands. However, though Exim is 8-bit clean, it is not a protocol converter, and it takes no steps to do anything special with messages received by this route. Consequently, this option is turned off by default.

accept_timeout

Type:	*time*
Default:	*0s*

This sets the timeout for accepting a non-SMTP message, that is, the maximum time that Exim waits when reading a message on the standard input. If the value is zero, it will wait for ever. This setting is overridden by the **-or** command option. The timeout for incoming SMTP messages is controlled by **smtp_receive_timeout**.

always_bcc

Type:	*boolean*
Default:	*false*

Exim adds a **To:** header to messages whose recipients are given on the command line when there is no **To:**, **Cc:**, or **Bcc:** in the message. In other cases of missing recipient headers, it just adds an empty **Bcc:** header to make the message conform with RFC 822. Setting **always_bcc** causes it to add an empty **Bcc:** in all cases. This can be helpful in conjunction with mailing list software that passes recipient addresses on the command line.

auto_thaw

Type:	*time*
Default:	*0s*

If this option is set to a non-zero time, a new delivery is attempted on frozen messages if this much time has passed since the message was frozen.

bi_command

Type:	*string*
Default:	*unset*

This option supplies the name of a command that is run when Exim is called with the **-bi** option (see chapter 5). The string value is just the command name, it is not a complete command line. If an argument is required, it must come from the **-oA** command line option.

check_log_inodes

Type:	*integer*
Default:	*0*

See **check_spool_space** below.

check_log_space

Type: *integer*
Default: *0*

See **check_spool_space** below.

check_spool_inodes

Type: *integer*
Default: *0*

See **check_spool_space** below.

check_spool_space

Type: *integer*
Default: *0*

The four **check_...** options allow for checking of disc resources before a message is accepted: **check_spool_space** and **check_spool_inodes** check the spool partition if either value is greater than zero, for example:

```
  check_spool_space = 10M
  check_spool_inodes = 100
```

The spool partition is the one which contains the directory defined by SPOOL_DIRECTORY in **Local/Makefile**.

check_log_space and **check_log_inodes** check the partition in which log files are written if either is greater than zero. These should be set only if **log_file_path** is set to point to a different partition to the spool directory.

If there is less space or fewer inodes than requested, Exim refuses to accept incoming mail. In the case of SMTP input this is done by giving a 452 temporary error response to the MAIL command. If ESMTP is in use and there was a SIZE parameter on the MAIL command, its value is added to the **check_spool_space** value, and the check is performed even if **check_spool_space** is zero, unless **no_smtp_check_spool_space** is set.

For non-SMTP input and for batched SMTP input, the test is done at start-up; on failure a message is written to stderr and Exim exits with a non-zero code, as it obviously cannot send an error message of any kind.

collapse_source_routes

Type: *boolean*
Default: *false*

If this option is set, then source-routed mail addresses are stripped down to their final components.

daemon_smtp_port

Type: *string*
Default: *unset*

This option specifies the numerical port number or the service name equivalent on which the daemon is to listen for incoming SMTP calls. It is overridden by **-oX** on the command line. If this option is not set, the service name ‘smtp’ is used.

daemon_smtp_service

Type: *string*
Default: *unset*

This option is a synonym for **daemon_smtp_port**.

debug_level

Type:	*integer*
Default:	*0*

This option sets the debug level, thus enabling it to be set when calling Exim from an MUA, but it is overridden by the use of **-d** on the command line.

delay_warning

Type:	*time-list*
Default:	*24h*

When a message is delayed, Exim sends a warning message to the sender at intervals specified by this option. If it is set to a zero, no warnings are sent. The data is a colon-separated list of times after which to send warning messages. Up to 10 times may be given. If a message has been on the queue for longer than the last time, the last interval between the times is used to compute subsequent warning times. For example, with

```
delay_warning = 4h:8h:24h
```

the first message is sent after 4 hours, the second after 8 hours, and subsequent ones every 16 hours thereafter. To stop warnings after a given time, set a huge subsequent time.

delay_warning_condition

Type:	*string*
Default:	*unset*

The string is expanded at the time a warning message might be sent. If the result of the expansion is a forced failure, an empty string, or a string matching any of '0', 'no' or 'false' (the comparison being done caselessly) then the warning message is not sent. For example

```
delay_warning_condition = "\
  ${if match{$h_precedence:}{(?i)bulk|list|junk}{no}{yes}}"
```

suppresses the sending of warnings about messages that have 'bulk' or 'list' or 'junk' in a **Precedence:** header. Note that the colon to terminate the header name is necessary because } may legally occur in header names.

deliver_load_max

Type:	*fixed-point*
Default:	*unset*

When this option is set, no message deliveries are ever done if the system load average is greater than its value, except for deliveries forced with the **-M** option. If **deliver_queue_load_max** is not set and the load gets this high during a queue run, the run is abandoned. There are some operating systems for which Exim cannot determine the load average (see chapter 1); for these this option has no effect.

deliver_queue_load_max

Type:	*fixed-point*
Default:	*unset*

If this option is set, its value is used to determine whether to abandon a queue run, instead of the value of **deliver_load_max**.

delivery_date_remove

Type:	*boolean*
Default:	*true*

Exim's transports have an option for adding a **Delivery-date:** header to a message when it is delivered – in exactly the same way as **Return-path:** is handled. **Delivery-date:** records the actual time of delivery. Such headers should not be present in incoming messages, and this option causes them to be removed, to avoid any problems that might occur when a delivered message is subsequently sent on to some other recipient.

dns_again_means_nonexist

Type: *domain-list*
Default: *unset*

DNS lookups give a 'try again' response for the DNS error 'non-Authoritive host found or SERVERFAIL'. This can cause Exim to keep trying to deliver a message, or to give repeated temporary errors to incoming mail. Sometimes the effect is caused by a badly set up nameserver and may persist for a long time. If a domain which exhibits this problem matches anything in **dns_again_means_nonexist** then it is treated as if it did not exist. This option should be used with care.

dns_check_names

Type: *boolean*
Default: *true*

This option causes Exim to check domain names for illegal characters before handing them to the DNS resolver, because some resolvers give temporary errors for bad names. If a domain name contains any illegal characters, a 'not found' result is forced. The check is done by matching the domain name against the regular expression specified by the **dns_check_names_pattern** option.

dns_check_names_pattern

Type: *string*
Default: *see below*

This option defines the regular expression that is used when the **dns_check_names** option is set. The default value is

```
dns_check_names_pattern =
  (?i)^(?>(?(1)\.|())[^\W_](?>[a-z0-9-]*[^\W_])?)+$
```

which permits only letters, digits, and hyphens in components, but they may not start or end with a hyphen.

dns_retrans

Type: *time*
Default: *0s*

The options **dns_retrans** and **dns_retry** can be used to set the retransmission and retry parameters for DNS lookups. Values of zero (the defaults) leave the system default settings unchanged. The first value is the time between retries, and the second is the number of retries. It isn't totally clear exactly how these settings affect the total time a DNS lookup may take. I haven't found any documentation about timeouts on DNS lookups; these parameter values are available in the external resolver interface structure, but nowhere does it seem to describe how they are used or what you might want to set in them.

dns_retry

Type: *integer*
Default: *0*

See **dns_retrans** above.

envelope_to_remove

Type: *boolean*
Default: *true*

Exim's transports have an option for adding an **Envelope-to:** header to a message when it is delivered – in exactly the same way as **Return-path:** is handled. **Envelope-to:** records the original recipient address in the envelope that caused the delivery. Such headers should not be present in incoming messages, and this option causes them to be removed, to avoid any problems that might occur when a delivered message is subsequently sent on to some other recipient.

errmsg_text

Type:	*string*
Default:	*unset*

If **errmsg_text** is set, its contents are included in the default error message immediately after ‘This message was created automatically by mail delivery software.’ It is not used if **errmsg_file** is set.

errmsg_file

Type:	*string*
Default:	*unset*

This option defines a template file containing paragraphs of text to be used for constructing the message which is sent by Exim in the case of a delivery failure. Details of the file’s contents are given in chapter 33. See also **warnmsg_file**.

errors_address

Type:	*string*
Default:	*"postmaster"*

The mail address to which Exim will send certain error reports. As the default is specified without a domain, it will be sent to the domain specified by the **qualify_recipient** option. If this address is specified with a domain, it must be a fully qualified domain.

errors_copy

Type:	*string-list*
Default:	*unset*

Setting this option causes Exim to send bcc copies of delivery failure reports that it generates to other addresses. The value is a colon-separated list of items; each item consists of a pattern and an address list, separated by white space. If the pattern matches the recipient of the delivery error report, the message is copied to the addresses on the list. The items are scanned in order, and once a matching one is found, no further items are examined. For example:

```
errors_copy = "spqr@mydomain   postmaster@mydomain :\
               rqps@mydomain   mailmaster@mydomain,\
                               postmaster@mydomain"
```

Each pattern can be a single regular expression, indicated by starting it with a circumflex; alternatively, either portion (local part, domain) can start with an asterisk, or the domain can be in any format that is acceptable as an item in a domain list, including a file lookup. A regular expression is matched against the entire (fully qualified) recipient; non-regular expressions must contain both a local part and domain, separated by @.

The address list is a string which is expanded, and must end up as a comma-separated list of addresses. It is used to construct a **Bcc:** header which is added to the error message. The expansion variables **local_part** and **domain** are set from the original recipient of the error message, and if there was any wildcard matching, the expansion variables **$0**, **$1**, etc. are set in the normal way.

errors_reply_to

Type:	*string*
Default:	*unset*

Exim’s delivery error messages contain the header

```
From: Mail Delivery System <Mailer-Daemon@${qualify_domain}>
```

(where string expansion notation is used to show a variable substitution). Experience shows that a large number of people reply to such messages. If the **errors_reply_to** option is set, a **Reply-to:** header is added. The option must specify the complete header body.

exim_group

Type: *string*
Default: *compile-time configured (can be unset)*

This option sets the gid under which Exim runs when it gives up root privilege. It is used only when **exim_user** is also set. Unless it consists entirely of digits, the string is looked up using **getgrnam()**, and failure causes a configuration error. See chapter 49 for a discussion of security issues.

exim_path

Type: *string*
Default: *see below*

This option specifies the path name of the Exim binary, which is used when Exim needs to re-exec itself. The default is set up to point to the file **exim** in the directory configured at compile time by the BIN_DIRECTORY setting. It is necessary to change **exim_path** if Exim is run from some other place.

exim_user

Type: *string*
Default: *compile-time configured (can be unset)*

This option sets the uid under which Exim runs when it gives up root privilege. Unless it consists entirely of digits, the string is looked up using **getpwnam()**, and failure causes a configuration error. If **exim_group** is not also supplied, the gid is taken from the result of **getpwnam()** if it is used. If the resulting uid is the root uid, it has the effect of unsetting this option. See chapter 49 for a discussion of security issues. Note that the ownership of the runtime configuration file is checked against the compile-time setting of this parameter, not what is set here.

extract_addresses_remove_arguments

Type: *boolean*
Default: *true*

According to Sendmail documentation, if any addresses are present on the command line when the **-t** option is used to build an envelope from a message's headers, they are removed from the recipients list. This is also how Smail behaves. However, it has been reported that some versions of Sendmail in fact add the argument addresses to the recipients list. By default Exim follows the documented behaviour, but if this option is set false it adds rather than removes argument addresses.

finduser_retries

Type: *integer*
Default: *0*

On systems running NIS or other schemes in which user and group information is distributed from a remote system, there can be times when **getpwnam()** and related functions fail, even when given valid data, because things time out. Unfortunately these failures cannot be distinguished from genuine 'not found' errors. If **finduser_retries** is set greater than zero, Exim will try that many extra times to find a user or a group, waiting for one second between tries.

forbid_domain_literals

Type: *boolean*
Default: *false*

If this option is set, the RFC 822 domain literal format is not permitted in addresses.

freeze_tell_mailmaster

Type:	*boolean*
Default:	*false*

On encountering certain errors, Exim freezes a message, which means that no further delivery attempts take place until an administrator thaws it. If this option is set, a message is sent to **errors_address** every time a message is frozen, unless the message is itself a delivery error message. (Without this exception there is the possibility of looping.) If several of the message's addresses cause freezing, only a single message is sent to the mail administrator. The reason(s) for freezing will be found in the message log.

gecos_name

Type:	*string*
Default:	*unset*

Some operating systems, notably HP-UX, use the 'gecos' field in the system password file to hold other information in addition to users' real names. Exim looks up this field for use when it is creating **Sender:** or **From:** headers. If either **gecos_pattern** or **gecos_name** are unset, the contents of the field are used unchanged, except that, if an ampersand is encountered, it is replaced by the user's login name with the first character forced to upper-case, since this is a convention that is observed on many systems.

When these options are set, **gecos_pattern** is treated as a regular expression that is to be applied to the field (again with & replaced by the login name), and if it matches, **gecos_name** is expanded and used as the user's name. Numeric variables such as **$1**, **$2**, etc. can be used in the expansion to pick up sub-fields that were matched by the pattern. In HP-UX, where the user's name terminates at the first comma, the following can be used:

```
gecos_pattern = "([^,]*)"
gecos_name = $1
```

gecos_pattern

Type:	*string*
Default:	*unset*

See **gecos_name** above.

headers_check_syntax

Type:	*boolean*
Default:	*false*

This option causes Exim to check the syntax of all headers that can contain lists of addresses (**Sender:**, **From:**, **Reply-to:**, **To:**, **Cc:**, and **Bcc:**) on all incoming messages (both local and SMTP). This is a syntax check only, to catch real junk such as

```
To: user@
```

Like the **headers_sender_verify** options, the rejection happens after the end of the data, but it is also controlled by **headers_checks_fail**; if that is unset, the message is accepted and a warning is written to the reject log.

If the message contains any headers starting with **Resent-** then it is that set of headers which is checked.

headers_checks_fail

Type:	*boolean*
Default:	*true*

If this option is true, failure of any header check (see below) causes the message to be rejected. If it is false, a warning message is written to the reject log.

headers_sender_verify

Type: *boolean*
Default: *false*

If this option is set with **sender_verify**, and the sending host matches **sender_verify_hosts**, Exim insists on there being at least one verifyable address in one of the **Sender:**, **Reply-to:**, or **From:** headers (which are checked in that order) on all incoming SMTP messages. If one cannot be found, the message is rejected, unless **headers_checks_fail** is unset, in which case a warning entry is written to the reject log.

If there are any headers whose names start with **Resent-**, then it is that set of headers which is checked. If there is more than one instance of a particular header, all of them are checked.

Unfortunately, because it has to read the message before doing this check, the rejection happens after the end of the data, and it is known that some mailers do not treat hard (5xx) errors correctly at this point – they keep the message on their spools and try again later, but that is their problem, though it does waste some resources.

headers_sender_verify_errmsg

Type: *boolean*
Default: *false*

This option acts like **headers_sender_verify**, except that it applies only to messages whose envelope sender is ‘<>’, that is, delivery error messages whose sender cannot be verified at the time the SMTP MAIL command is received.

helo_accept_junk_hosts

Type: *host-list*
Default: *unset*

Exim checks the syntax of HELO and EHLO commands for incoming SMTP mail, and gives an error response for invalid data. Unfortunately, there are some SMTP clients that send syntactic junk. They can be accommodated by setting this option.

helo_strict_syntax

Type: *boolean*
Default: *false*

Because so many systems have been found to use underscores in the names they send in the SMTP HELO command, Exim by default permits them, though it is not in fact legal to use underscores in domain names. If **helo_strict_syntax** is set, underscores are not permitted in HELO or EHLO commands.

helo_verify

Type: *host-list*
Default: *unset*

The RFCs mandate that a server must not reject a message because it doesn’t like the HELO or EHLO command. However, some sites like to be stricter. If **helo_verify** is set, Exim checks each incoming call from any host that matches it, and accepts the call only if:

- A HELO or EHLO command is received;

and

- The host name given in that command either:

  (i) is an IP literal matching the calling address of the host (the RFCs specifically allow this), or

  (ii) matches the host name that Exim obtains by doing a reverse lookup of the calling host address, or

(iii) when looked up using **gethostbyname()** yields the calling host address.

If no HELO or EHLO is given, MAIL commands are rejected; if a bad HELO or EHLO is given, it is rejected with a 550 error. Rejections are logged in the main and reject logs.

hold_domains

Type:	*domain-list*
Default:	*unset*

This option allows mail for particular domains to be held on the queue manually. The option is overridden if a message delivery is forced with the **-M** or **-qf** options. Otherwise, if a domain matches an item in **hold_domains**, no routing or delivery for that address is done, and it is deferred every time the message is looked at.

This option is intended as a temporary operational measure for delaying the delivery of mail while some problem is being sorted out, or some new configuration tested. It does not override Exim's message clearing away code, which removes messages from the queue if they have been there longer than the longest retry time in any retry rule. If you want to hold messages for longer than the normal retry times, insert a dummy retry rule with a long retry time.

host_accept_relay

Type:	*host-list*
Default:	*unset*

An MTA is said to *relay* a message if it receives it from some host and delivers it directly to another host as a result of a remote address contained in it. Expanding a local address via an alias or forward file and then passing the message to another host is not relaying, but a re-direction as a result of the 'percent hack' is.

Two kinds of relaying exist, which might be termed 'incoming' and 'outgoing'. A host which is acting as a gateway or an MX backup is concerned with incoming relaying from arbitrary hosts to a specific set of domains. A host which is acting as a smart host for a number of clients is concerned with outgoing relaying from those known clients to the Internet at large. Often the same host is fulfilling both functions, but they are in principle entirely independent, and are therefore controlled by separate Exim options.

Incoming relaying is controlled by restricting the domains to which an arbitrary host may send via the local host; this is done by setting **relay_domains**. Outgoing relaying is controlled by restricting the set of hosts which may send via the local host to an arbitrary domain, by setting **host_accept_relay**.

A check for unwanted relaying is made on the domains of recipient addresses in messages received from other hosts. If the 'percent hack' is in use, the test is applied to the domains of the transformed addresses. The check is done at the time of the RCPT command in the SMTP dialogue.

If the domain in a recipient address matches **local_domains** or **relay_domains**, or if **relay_domains_include_local_mx** is set and the domain has an MX record pointing to the local host, the address is always accepted (at least as far as this check is concerned – a subsequent verification check might fail it). This is the case of an incoming message to a local domain or an incoming relay to a permitted domain.

Otherwise this is a case of outgoing relaying, and the address is accepted only if the host is permitted to relay to arbitrary domains, as specified by **host_accept_relay**, and, if **sender_address_relay** is set, the sender's address from the MAIL command matches it. In other words, both the host and the sender address must be acceptable. However, some installations are prepared to accept relaying on the basis of either the host or the sender address, and to permit this, **relay_match_host_or_sender** can be set. As it is very easy to forge sender addresses, this option should be used with caution.

Chapter 40 contains further discussion of relay control.

host_lookup

Type: *host-list*
Default: *unset*

Exim does not look up the name of a calling host from its IP address unless it is required to compare against some host list, or **helo_verify** is set, or the address matches this option (which normally contains IP addresses rather than host names, since the presence of names in itself implies a DNS lookup). The default configuration file contains

```
host_lookup = 0.0.0.0/0
```

which causes a lookup to happen for all hosts. If the expense of these lookups is felt to be too great, the setting can be changed or removed. However, Exim always does a lookup if the domain name quoted in a HELO or EHLO command is the local host's own name or any of its local mail domains.

host_reject

Type: *host-list*
Default: *unset*

If this option is set, incoming SMTP calls from the hosts listed (possibly also qualified by an RFC 1413 identification) are rejected as soon as the connection is made. See chapter 40 for more details.

host_reject_recipients

Type: *host-list*
Default: *unset*

If this option is set, all recipients in incoming SMTP calls from the hosts listed, possibly also qualified by an RFC 1413 identification, are rejected. Chapter 40 contains details of this facility, which differs from **host_reject** only in the point in the SMTP dialogue at which the rejection occurs.

hosts_treat_as_local

Type: *domain-list*
Default: *unset*

If this option is set, any host names that match the domain list are treated as if they were the local host when Exim is scanning host lists obtained from MX records, and also at other times when it is checking whether a host to which a message has been routed is the local host. If it is required that the matching host names also be treated as local domains for mail delivery, they must appear in **local_domains** as well as in this option.

See also the **allow_localhost** option in the **smtp** transport. Both these options are needed in a setup with different hosts for incoming and outgoing mail if the resulting system is used for MX backup.

ignore_errmsg_errors

Type: *boolean*
Default: *false*

If this option is set, failed addresses in error reports (that is, messages whose senders are '<>') are discarded (with a log entry). The default action is to freeze such messages for human attention.

ignore_errmsg_errors_after

Type: *time*
Default: *0s*

This option, if it is set to a non-zero time, acts as a delayed version of **ignore_errmsg_errors**, which must be unset for this option to take effect. If an error message is frozen because of delivery failure, then once the given time has elapsed after the freezing took place, the message is unfrozen at the next queue run. If delivery fails again, the error message is discarded. This makes it possible to keep failed error messages around for a shorter time than the normal maximum retry time.

ignore_fromline_hosts

Type:	*host-list*
Default:	*unset*

Some broken SMTP clients insist on sending a UUCP-like 'From' line before the headers of a message. By default this is treated as the start of the message's body, which means that any following headers are not recognized as such. Exim can be made to ignore it by setting **ignore_fromline_hosts** to match those hosts that insist on sending it. If the sender is actually a local process rather than a remote host, and is using **-bs** to inject the messages, then **ignore_fromline_local** can be set to deal with this case.

ignore_fromline_local

Type:	*boolean*
Default:	*false*

See **ignore_fromline_hosts** above.

keep_malformed

Type:	*time*
Default:	*4d*

This option specifies the length of time to keep messages whose spool files have been corrupted in some way. This should, of course, never happen. At the next attempt to deliver such a message, it gets removed. The incident is logged.

kill_ip_options

Type:	*boolean*
Default:	*true*

IP packets can contain options which are *source routing* data that enables one host to pretend to be another. (Don't confuse IP source routing with source-routed mail addresses, which are something entirely different.) IP source routing is an obvious security risk, and many sites lock out such packets in their routers. Also, some operating systems are able to disable IP source routing at the kernel level.

If Exim receives an SMTP call with IP options set, it logs the options if **log_ip_options** is set. Then, if **refuse_ip_options** is set, it drops the call; otherwise, if **kill_ip_options** is set, it unsets the options on the outgoing socket and attempts to continue. To read the IP options, **getsockopt()** is used. On some versions of SunOS 4.1 this causes system crashes. There is a patch that fixes this problem, but it can be avoided by setting all three Exim options false.

local_domains

Type:	*domain-list*
Default:	*see below*

This specifies a list of domains which are recognized as 'local', that is, their delivery is handled in a special way by this MTA using directors rather than routers. If this option is not set, it defaults to the value of **qualify_recipient**.

The name of the local host is not by default recognized as a local mail domain; either it must be included in **local_domains**, or the **local_domains_include_host** option must be set. If you want to accept mail addressed to your host in RFC 822 domain literal format, then **local_domains** must also include the appropriate 'domains', consisting of IP addresses enclosed in square brackets. The **local_domains_include_host_literals** option can be set to add all IP addresses automatically.

It is possible to specify *no* local domains by specifying no data for this option, for example,

```
local_domains =
```

If there are very many local domains, then they can be stored in a file and looked up whenever this string is searched. See the discussion of domain lists in section 7.12.

local_domains_include_host

Type:	*boolean*
Default:	*false*

If this option is set, the value of **primary_hostname** is added to the value of **local_domains**, unless it is already present. This makes it possible to use the same configuration file on a number of different hosts. The same effect can be obtained by including the conventional item ‘@’ (which matches the primary host name) in **local_domains**.

local_domains_include_host_literals

Type:	*boolean*
Default:	*false*

If this option is set and **local_interfaces** is unset, the IP addresses of all the interfaces on the local host, with the exception of 127.0.0.1 (and ::1 on IPv6 systems), are added to the value of **local_domains**, in domain literal format, that is, as strings enclosed in square brackets. If **local_interfaces** is set, then only those addresses it contains (again excluding 127.0.0.1 and ::1) are used.

local_interfaces

Type:	*string-list*
Default:	*unset*

The string must contain a list of IP addresses, in dotted-quad format for IPv4 addresses, or in colon-separated format (with colons doubled) for IPv6 addresses. These are used for two different purposes:

- When a daemon is started to listen for incoming SMTP calls, it listens only on the interfaces identified here, that is, it calls **bind()** for these interfaces only. An error occurs if it is unable to bind a listening socket to any interface.
- Only the IP addresses listed here are taken as the local host’s addresses when routing mail and checking for mail loops.

If **local_interfaces** is unset, the daemon issues a generic **listen()** that accepts incoming calls from any interface, and it also gets a complete list of available interfaces and treats them all as local when routing mail. On most systems the default action is what is wanted. However, some systems set up large numbers of virtual interfaces in order to provide many different virtual web servers. In these cases **local_interfaces** can be used to restrict SMTP traffic to one or two interfaces only. See also **hosts_treat_as_local**.

localhost_number

Type:	*string*
Default:	*unset*

Exim’s message ids are normally unique only within the local host. If uniqueness among a set of hosts is required, then each host must set a different value for the **localhost_number** option. The string is expanded immediately after reading the configuration file (so that a number can be computed from the host name, for example) and the result of the expansion must be a number in the range 0–255. This is available in subsequent string expansions via the variable **$localhost_number**. The final two characters of the message id, instead of just being a sequence count of the number of messages received by one process in one second, are the base 62 encoding of

<*sequence count*> `* 256 +` <*local host number*>

This reduces the possible range of the sequence count to 0–14. If the count ever reaches 14 in a receiving process, a delay of one second is imposed to allow the clock to tick, thereby allowing the count to be reset to zero.

locally_caseless

Type: *boolean*
Default: *true*

For most Unix systems it is desirable that local parts of local mail addresses be treated in a case-independent manner, since most users expect that mail to **OBailey** and **obailey**, for example, will end up in the same mailbox. By default, Exim lower-cases local local parts at the start of processing them, on the assumption that account names in the password file are in lower-case.

For installations that want to draw case distinctions, this option is provided. When turned off, local local parts are handled verbatim during delivery. If there are names containing upper case letters in the password file, the most convenient way to provide for caseless mail delivery is to set up a **smartuser** director as the first director, and to make it do a lowercased lookup of the local part, in order to translate it to the correctly cased version, using the **new_address** option.

log_all_parents

Type: *boolean*
Default: *false*

This option applies to deliveries of local addresses, where the original envelope address may be converted by (for example) an alias file into a 'child' address which might itself be an alias. Thus in general there can be a chain of several addresses between the original one and the address to which the actual delivery is made. By default Exim logs the final address, followed by the original address in angle bracket.

Turning **log_all_parents** on causes all intermediate addresses between the original envelope address and the final delivery address to be included in delivery log lines in parentheses after the first address. Without this, intermediate addresses are not included, except that if the final delivery is to a pipe or file or autoreply, the immediately preceding parent address is listed.

log_arguments

Type: *boolean*
Default: *false*

Setting this option causes Exim to write the arguments with which it was called to the main log. This is a debugging feature, added to make it easy to find out with what arguments certain MUAs call **/usr/lib/sendmail**. The logging does not happen if Exim has given up root privilege because it was called with the **-C** or **-D** options.

log_file_path

Type: *string*
Default: *compile-time configured (may be unset)*

This option sets the path which is used to determine the names of Exim's log files. The string is expanded, so it can contain, for example, references to the host name. After expansion it must contain the string '%s' somewhere within it; this will be replaced with one of the strings 'main', 'panic', 'process', or 'reject' to form the final file name. For example,

```
log_file_path = /var/log/${primary_hostname}/exim_%slog
```

If this path string is fixed at your installation (contains no expansion variables) it is recommended that you do not set this option in the configuration file, but instead supply the path using LOG_FILE_PATH in **Local/Makefile** so that it is available to Exim for logging errors detected early on – in particular failure to read the configuration file.

If no specific path is set for the log files, they are written in a sub-directory called **log** in Exim's spool directory.

log_ip_options

Type:	*boolean*
Default:	*true*

See **kill_ip_options** above.

log_level

Type:	*integer*
Default:	*5*

This controls the amount of data written to the main log and to the individual message logs (see section 45.7). The higher the number, the more is written. At present a value of 6 or higher causes all possible messages to appear.

log_received_recipients

Type:	*boolean*
Default:	*false*

When this option is set, the recipients of a message are listed in the main log as soon as the message is received. The list appears at the end of the log line that is written when a message is received, preceded by the word 'for'. The addresses are listed after they have been qualified, but before any rewriting has taken place.

log_received_sender

Type:	*boolean*
Default:	*false*

If this option is set, the unrewritten original sender of a message is added to the end of the log line that records the message's arrival, after the word 'from' (before the recipients if **log_received_recipients** is also set).

log_refused_recipients

Type:	*boolean*
Default:	*false*

If this option is set, an entry is written in the main and reject logs for each recipient that is refused for policy reasons. Otherwise cases where all recipients are to be refused just cause a single log entry for the message.

log_rewrites

Type:	*boolean*
Default:	*false*

This option causes all address rewriting to get logged, as an aid to debugging rewriting rules.

log_smtp_confirmation

Type:	*boolean*
Default:	*false*

This option causes the response to the final '.' in the SMTP dialog for outgoing messages to be added to delivery log lines in the form 'C="<*text*>"'. A number of MTAs (including Exim from release 1.60) return an identifying string in this response, so logging this information allows messages to be tracked more easily. This global option applies to all SMTP transports.

log_smtp_connections

Type:	*boolean*
Default:	*false*

This option turns on more verbose logging of incoming SMTP connections, at log level 4. This does not apply to batch SMTP, but it does apply to SMTP connections from local processes that use the **-bs** option, including incoming calls using **inetd**. A log line is written whenever a connection is established or closed. If a connection is dropped in the middle of a message, a log line is always written, but otherwise nothing is written at the start and end of connections unless **log_smtp_ connections** is set.

log_smtp_syntax_errors

Type:	*boolean*
Default:	*false*

If this option is set, syntax errors in incoming SMTP commands are logged at level 4. An unrecognized command is treated as a syntax error. For an external connection, the host identity is given; for an internal connection using **-bs** the sender identification (normally the calling user) is given.

log_subject

Type:	*boolean*
Default:	*false*

This option causes a message's subject to be included in the arrival log line, in the form 'T="<*subject text*>"'. T stands for 'topic' (S is already used for 'size').

lookup_open_max

Type:	*integer*
Default:	*25*

This option limits the number of simultaneously open lookup files. Exim normally keeps files open during directing and routing, since often the same file is required several times. This limit applies only to those lookup types which use regular files, namely lsearch, dbm, and cdb. If the limit is reached, Exim closes the least recently used file. Note that if you are using the NDBM library, it actually opens two files for each logical DBM database, though it still counts as one for the purposes of **lookup_open_max**. If you are getting 'too many open files' errors with NDBM, you need to reduce the value of **lookup_open_max**.

max_user_name_length

Type:	*integer*
Default:	*0*

Some operating systems are broken in that they truncate the argument to **getpwnam()** to eight characters, instead of returning 'no such user'. If this option is set greater than zero, any attempt to call **getpwnam()** with an argument that is longer behaves as if **getpwnam()** failed.

message_body_visible

Type:	*integer*
Default:	*500*

This option specifies how much of a message's body is to be included in the **message_body** expansion variable.

message_filter

Type:	*string*
Default:	*unset*

This option specifies a filter file which is applied to all messages before any routing or directing is done. This is called the 'system message filter'. If the filter generates any deliveries to files or pipes, or any new mail messages, then the appropriate **message_filter_..._transport** option(s) must be set, to define which transports are to be used. Details of this facility are given in chapter 41.

message_filter_directory_transport

Type: *string*
Default: *unset*

This sets the name of the transport driver that is to be used when the **save** command in a system message filter specifies a path ending in '/', implying delivery of each message into a separate file in some directory.

message_filter_directory2_transport

Type: *string*
Default: *unset*

This sets the name of the transport driver that is to be used when the **save** command in a system message filter specifies a path ending in '//'. The reason for having both **message_filter_directory** and **message_filter_directory2** is to allow for the rare circumstance in which both maildir and non-maildir format delivery is required.

message_filter_file_transport

Type: *string*
Default: *unset*

This sets the name of the transport driver that is to be used when the **save** command in a system message filter results in a path not ending in '/'.

message_filter_group

Type: *string*
Default: *unset*

This option sets the gid under which the system message filter is run. The **seteuid()** or **setresuid()** function must be available in the operating system for a temporary change to be possible. If the filter generates any pipe, file, or reply addresses, the gid under which the filter is run is used when delivering to them. Unless the string consists entirely of digits, it is looked up using **getgrnam()**, and failure causes a configuration error. If the option is not set, and either **message_filter_user** is unset or consists entirely of digits, the gid is not changed when running the filter. Otherwise the group is taken from the result of **getpwnam()**.

message_filter_pipe_transport

Type: *string*
Default: *unset*

This sets the name of the transport driver that is to be used when a **pipe** command is used in a system message filter.

message_filter_reply_transport

Type: *string*
Default: *unset*

This sets the name of the transport driver that is to be used when a **mail** command is used in a system message filter.

message_filter_user

Type: *string*
Default: *unset*

This option sets the uid under which the system message filter is run. The **seteuid()** or **setresuid()** function must be available in the operating system for a temporary change to be possible. If the filter generates any pipe, file, or reply addresses, the uid under which the filter is run is used when delivering to them. Unless it consists entirely of digits, the string is looked up using **getpwnam()**, and failure causes a configuration error. If the option is not set, the uid is not changed from the Exim user (or root if there is no Exim user) when running the system filter.

message_id_header_text

Type: *string*
Default: *unset*

If this variable is set, the string is expanded and used to augment the text of the **Message-id:** header that Exim creates if an incoming message does not have one. The text of this header is required by RFC 822 to take the form of an address. By default, Exim uses its internal message id as the local part, and the primary host name as the domain. If this option is set, it is expanded and provided the expansion does not yield an empty string, is is inserted into the header immediately before the @, separated from the internal message id by a dot. Any characters that are illegal in an address are automatically converted into hyphens. This means that constructions like **${tod_log}** can be used, as the spaces and colons will become hyphens.

message_size_limit

Type: *integer*
Default: *0*

This option limits the maximum size of message that Exim will process. Zero means no limit. It should be set somewhat larger than **return_size_limit** if the latter is non-zero. Incoming SMTP messages are failed with a 552 error if the limit is exceeded; locally-generated messages either get a stderr message or a delivery failure message to the sender, depending on the **-oe** setting, in the normal way. Rejection of an oversized message is logged in both the main and the reject logs. See also the generic transport option **message_size_limit**, which limits the size of message that an individual transport can process.

message_size_limit_count_recipients

Type: *boolean*
Default: *false*

If this option is set, then the value of **message_size_limit** is a maximum for the size of a message times the number of envelope recipients it has. For example, if **message_size_limit** is set to 10M, then a message with 4 recipients can be no bigger than 2.5M, and a message with 100 recipients is limited to around 100K.

never_users

Type: *string-list*
Default: *unset*

Local mail deliveries are run in processes that are setuid to the recipient. However, it is usually desirable to lock out root from this, as a safety precaution. If a message is to be delivered locally as any of the users on the **never_users** list, the process is run as ‘nobody’ instead (see **nobody_user** below). A common example is

```
never_users = root:daemon:bin:exim
```

This option overrides the **pipe_as_creator** option of the **pipe** transport driver. If Exim is unable to find a uid for ‘nobody’, it panics.

nobody_group

Type: *string*
Default: *unset*

This specifies the group to use when a process is to be run as ‘nobody’. If it is unset, the value of the ‘nobody’ user’s default group is used.

nobody_user

Type: *string*
Default: *unset*

This specifies the user to use when a process is to be run as 'nobody'. If it is unset, Exim looks up the user 'nobody' using **getpwnam()**. If this fails, Exim panics, writing a message to the panic log and exiting immediately.

percent_hack_domains

Type: *domain-list*
Default: *unset*

The 'percent hack' is the convention whereby a local part containing a percent sign is re-interpreted as a remote address, with the percent replaced by @. This is sometimes called 'source routing', though that term is also applied to RFC 822 addresses that begin with an @ character. If this option is set, Exim implements the percent facility for those local domains listed, but no others. The option can be set to '*' to allow the percent hack for all local domains.

If options are set to control message relaying from incoming SMTP envelopes, they are also applied to relaying that is requested via the 'percent hack'. See section 40.4.

perl_at_start

Type: *boolean*
Default: *false*

This option is available only when Exim is built with an embedded Perl interpreter. See chapter 10 for details of its use.

perl_startup

Type: *string*
Default: *unset*

This option is available only when Exim is built with an embedded Perl interpreter. See chapter 10 for details of its use.

pid_file_path

Type: *string*
Default: *compile-time configured (may be unset)*

This option sets the path which is used to determine the name of the file to which the Exim daemon writes its process id. The string is expanded, so it can contain, for example, references to the host name. After expansion it must contain the string '%s' somewhere within it; this will be replaced by the null string or a non-standard port number to form the final file name. For example,

```
pid_file_path = /var/log/${primary_hostname}/exim%s.pid
```

If no specific path is set for the file, it is written in Exim's spool directory.

preserve_message_logs

Type: *boolean*
Default: *false*

If this option is set, message log files are not deleted when messages are completed. Instead, they are moved to a sub-directory of the spool directory called **msglog.OLD**, where they remain available for statistical or debugging purposes. This is a dangerous option to set on systems with any appreciable volume of mail. Use with care!

primary_hostname

Type: *string*
Default: *see below*

This specifies the name of the current host. This is used in the HELO command for outgoing SMTP messages, and as the default for **qualify_domain**. If it is not set, Exim calls **uname()** to find it. If this fails, Exim panics and dies. If the name returned by **uname()** contains only one component, Exim passes it to **gethostbyname()** in order to obtain the fully qualified version.

print_topbitchars

Type: *boolean*
Default: *false*

By default, Exim considers only those characters whose codes lie in the range 32–126 to be printing characters. In a number of circumstances (for example, when writing log entries) non-printing characters are converted into escape sequences, primarily to avoid messing up the layout. If **print_topbitchars** is set, code values of 128 and above are also considered to be printing characters.

prod_requires_admin

Type: *boolean*
Default: *true*

The **-M**, **-R**, and **-q** command-line options require the caller to be an admin user unless **prod_requires_admin** is set false. See also **queue_list_requires_admin**.

prohibition_message

Type: *string*
Default: *unset*

This option adds a site-specific message to the error response that is sent when an SMTP command fails for policy reasons, for example if the sending host is in a host reject list. Details of this facility are given in chapter 40.

qualify_domain

Type: *string*
Default: *see below*

This specifies the domain name that is added to any sender addresses that do not have a domain qualification. It also applies to recipient addresses if **qualify_recipient** is not set. Such addresses are accepted by default only for locally-generated messages – messages from external sources must always contain fully qualified addresses, unless the sending host matches one of the **receiver_unqualified** or **sender_unqualified** options. If **qualify_domain** is not set, it defaults to the **primary_hostname** value.

qualify_recipient

Type: *string*
Default: *see below*

This specifies the domain name that is added to any recipient addresses that do not have a domain qualification. Such addresses are accepted by default only for locally-generated messages – messages from external sources must always contain fully qualified addresses, unless the sending host matches one of the **receiver_unqualified** or **sender_unqualified** options (see below). If **qualify_recipient** is not set, it defaults to the **qualify_domain** value.

queue_list_requires_admin

Type: *boolean*
Default: *true*

The **-bp** command-line option requires the caller to be an admin user unless **queue_list_requires_admin** is set false. Otherwise, only messages that the caller submitted are displayed. See also **prod_requires_admin**.

queue_only

Type: *boolean*
Default: *false*

If **queue_only** is set (which is equivalent to the **-odq** command line option), a delivery process is not automatically started whenever a message has been received. Instead, the message waits on the queue for the next queue run. Even if **queue_only** is false, incoming SMTP messages may not get delivered immediately if a lot of them arrive at once – see the **queue_only_load** and **smtp_accept_queue** options.

queue_only_file

Type: *string*
Default: *unset*

This option can be set to a colon-separated list of absolute path names, each one optionally preceded by 'remote' or 'smtp'. When it is receiving a message, Exim tests for the existence of each listed path using a call to **stat()**, and if this succeeds, the corresponding queuing option is set. If there is no prefix to the path, **queue_only** is set; 'remote' corresponds to **queue_remote** and 'smtp' to **queue_smtp**. So, for example,

```
  queue_only_file = remote/some/file
```

causes Exim to behave as if **queue_remote** were set to '*' whenever **/some/file** exists.

queue_only_load

Type: *fixed-point*
Default: *unset*

If the system load average is higher than this value, all incoming messages are queued, and no automatic deliveries are started. If this happens during local or remote SMTP input, all subsequent messages on the same connection are queued. Deliveries will subsequently be performed by queue running processes, unless the load is higher than **deliver_load_max**. There are some operating systems for which Exim cannot determine the load average (see chapter 1); for these this option has no effect. See also **smtp_accept_queue** and **smtp_load_reserve**.

queue_remote_domains

Type: *domain-list*
Default: *unset*

This option lists domains for which local delivery is not immediately required. It is checked against the domains supplied in the incoming addresses, before any widening is done (because that is part of routing). The **-odqr** option is equivalent to setting **queue_remote_domains** to '*'. A delivery process is started whenever a message is received, but only local addresses are handled, and only local deliveries take place. All remote deliveries wait until the next queue run. See also **queue_smtp_domains**, which is subtly different.

queue_run_in_order

Type: *boolean*
Default: *false*

If this option is set, queue runs happen in order of message arrival instead of in an arbitrary order.

queue_run_max

Type:	*integer*
Default:	*5*

This controls the maximum number of queue-running processes that an Exim daemon will run simultaneously. This does not mean that it starts them all at once, but rather that if the maximum number are still running when the time comes to start another one, it refrains from starting it. This can happen with very large queues and/or very sluggish deliveries. This option does not, however, interlock with other processes, so additional queue-runners can be started by other means, or by killing and restarting the daemon.

queue_smtp_domains

Type:	*domain-list*
Default:	*unset*

When this option is set, a delivery process is started whenever a message is received, directing and routing is performed, and local deliveries take place. However, if any SMTP deliveries are required for domains that match **queue_smtp_domains**, they are not immediately delivered, but instead the message waits on the queue for the next queue run. Since routing of the mes sage has taken place, Exim knows to which remote hosts it must be delivered, and so when the queue run happens, multiple messages for the same host are delivered over a single SMTP connection. This option is checked against the domains supplied in the incoming addresses, before any widening is done (because that is part of routing). The **-odqs** command line option causes all SMTP deliveries to be queued in this way, and is equivalent to setting **queue_smtp_domains** to '∗'. See also **queue_remote_domains**, which is subtly different.

rbl_domains

Type:	*string-list*
Default:	*unset*

This option is part of the support for Realtime Blocking Lists (RBL), details of which are given in chapter 40. It can be set to a colon-separated list of DNS RBL domains in which to look up the inverted IP address of a calling host. An RBL domain name may be followed by '/warn' or '/reject' to specify what is to be done if the host is found, for example:

```
rbl_domains = dul.maps.vix.com/warn : rbl.maps.vix.com/reject
```

If neither is present, the action is controlled by the setting of **rbl_reject_recipients**. When a lookup succeeds, and the action is 'reject', mail from the host is blocked by refusing all recipients, except those listed in **recipients_reject_except**. When the action is 'warn', the incident is just logged, and a header may be added to the message (see **rbl_warn_header**). If a lookup times out or otherwise fails to give a decisive answer, the address is not blocked (by that entry in the list). When blocking occurs, an associated TXT record is looked up in the DNS, and if it exists, its contents are returned as part of the 550 rejection message.

rbl_hosts

Type:	*host-list*
Default:	∗

This option specifies the set of hosts for which RBL checking is to be performed when **rbl_domains** is set. The default matches all hosts. The normal usage of this option is to specify exceptions to RBL checking by means of negated items in the host list.

rbl_log_headers

Type:	*boolean*
Default:	*false*

When this option is set, the headers of each message received from a host that matches an RBL domain are written to the reject log. This can occur only if the recipients of the message are not rejected, that is, if the RBL check is configured to warn only.

rbl_log_rcpt_count

Type:	*boolean*
Default:	*false*

When this option is set and **rbl_reject_recipients** is false, the number of RCPT commands for each message received from a host that is in the RBL is written to the reject log. This may be greater than the number of valid recipients in the message.

rbl_reject_recipients

Type:	*boolean*
Default:	*true*

This option controls the action taken when a remote host is found in an RBL domain that has neither '/warn' nor '/reject' following it. The default value specifies rejection.

rbl_warn_header

Type:	*boolean*
Default:	*true*

When this option is set and a message from an RBL-matching host is not rejected, an **X-RBL-Warning:** header is added. The header contains the contents of the DNS TXT record, if one was found. Scanning of further RBL domains continues, which means that more than one **X-RBL-Warning:** header may be added to a message.

received_header_text

Type:	*string*
Default:	*see below*

This string defines the contents of the **Received:** message header that is added to each message, except for the timestamp, which is automatically added on at the end, preceded by a semicolon. The string is expanded each time it is used, and the default is:

```
received_header_text = "Received: \
    ${if def:sender_rcvhost {from ${sender_rcvhost}\n\t}\
    {${if def:sender_ident {from ${sender_ident} }}\
    ${if def:sender_helo_name {(helo=${sender_helo_name})\n\t}}}}\
    by ${primary_hostname} \
    ${if def:received_protocol {with ${received_protocol}}} \
    (Exim ${version_number} #${compile_number})\n\t\
    id ${message_id}
    ${if def:received_for {\n\tfor $received_for}}"
```

The use of conditional expansions ensures that this works for both locally generated messages and messages received from remote hosts, giving header lines such as the following:

```
Received: from scrooge.carol.book ([240.1.12.25] ident=root)
        by marley.carol.book with smtp (Exim 1.90 #1)
        id E0tS3Ga-0005C5-00
        for cratchit@dickens.book; Mon, 25 Dec 1995 14:43:44 +0000
Received: by scrooge.carol.book with local (Exim 1.90 #1)
        id E0tS3GW-0005C2-00; Mon, 25 Dec 1995 14:43:41 +0000
```

Note the automatic addition of the date and time in the required format.

received_headers_max

Type: *integer*
Default: *30*

When a message is to be delivered, the number of **Received:** headers is counted, and if it is greater than this parameter, a mail loop is assumed to have occurred, the delivery is abandoned, and an error message is generated. This applies to both local and remote deliveries. Earlier versions of Exim did this test only for remote deliveries, but because local deliveries (as Exim sees them) may in fact still cause a message to be transported to a remote host, it was changed.

receiver_try_verify

Type: *boolean*
Default: *false*

See **receiver_verify**.

receiver_unqualified_hosts

Type: *host-list*
Default: *unset*

This option lists those hosts from which Exim is prepared to accept unqualified receiver addresses. The addresses are made fully qualified by the addition of the **qualify_recipient** value. Typically the hosts are local ones, but if you want to imitate the behaviour of mailers that accept unqualified addresses from anywhere, specify

```
receiver_unqualified_hosts = *
```

receiver_verify

Type: *boolean*
Default: *false*

When this option is set, the addresses of recipients received from a remote host are verified as they are received, provided the sending host matches **receiver_verify_hosts**, the incoming address matches **receiver_verify_addresses**, and the sender address matches **receiver_verify_senders**, if either of the last two are set.

If an address is invalid, an incoming SMTP call gets an error response to the RCPT command. If an address cannot immediately be verified, a temporary error code is given. The **receiver_try_verify** option is less severe: it operates in the same way, except that an address is accepted if it cannot immediately be verified. Verification failures are logged.

receiver_verify_addresses

Type: *address-list*
Default: *unset*

If set, this option restricts receiver verification to those addresses it matches. The option is inspected only if **receiver_verify** or **receiver_try_verify** is set.

receiver_verify_hosts

Type: *host-list*
Default: *

See **receiver_verify** above.

receiver_verify_senders

Type: *address-list*
Default: *unset*

This option, if set, allows receiver verification to be conditional upon the sender. It is inspected only if **receiver_verify** or **receiver_try_verify** is set.

If the null sender is required in the list of addresses, then it must not be the last item, as a null last item in a list is ignored. It is best placed at the start of the list. For example, to restrict receiver verification to messages with null senders and senders in the **.com** and **.org** domains, you could have

```
receiver_verify
receiver_verify_senders = :*.com:*.org
```

If the null sender is the only entry required, then the list should consist of a single colon.

recipients_max

Type: *integer*
Default: *0*

If this is set greater than zero, it specifies the maximum number of recipients for any message. This applies to the original list of recipients supplied with the message. SMTP messages get a 452 response for all recipients over the limit; earlier recipients are delivered as normal. Non-SMTP messages with too many recipients are failed, and no deliveries are done. Note that the RFCs specify that an SMTP server should accept at least 100 RCPT commands in a single message.

recipients_max_reject

Type: *boolean*
Default: *false*

If this option is set true, then Exim rejects SMTP messages containing too many recipients by giving 552 errors to the surplus RCPT commands, and a 554 error to the eventual DATA command. Otherwise (the default) it gives a 452 error to the surplus RCPT commands and accepts the message on behalf of the initial set of recipients. The remote server should then re-send the message for the remaining recipients at a later time.

recipients_reject_except

Type: *address-list*
Default: *unset*

This option lists recipient addresses which are exceptions to any policy for recipient rejection, that is, as a result of **sender_reject_recipients**, etc. This option is entirely independent of any checks for unwanted message relaying. However, it does interact with the RBL options.

refuse_ip_options

Type: *boolean*
Default: *true*

See **kill_ip_options** above.

relay_domains

Type: *domain-list*
Default: *unset*

See **host_accept_relay** and related options above.

relay_domains_include_local_mx

Type: *boolean*
Default: *false*

This option permits any host to relay to any domain that has an MX record pointing at the local host. It causes any domain with an MX record pointing at the local host to be treated as if it were in **relay_domains**. See **host_accept_relay** above. **Warning:** Turning on this option opens your server to the possibility of abuse in that anyone with access to a DNS zone can list your server in a secondary MX record as a backup for their domain without your permission. This is not a huge exposure because firstly, it requires the cooperation of a hostmaster to set up, and secondly, since their mail is passing through your server, they run the risk of your noticing and (for example) throwing all their mail away.

relay_match_host_or_sender

Type: *boolean*
Default: *false*

By default, if relaying controls are specified on both the remote host and the sender address, a message is accepted only if both conditions are met. If **relay_match_host_or_sender** is set, then either condition is good enough. It does not make sense to set this option without setting **sender_address_relay**, since if that option is unset it matches all senders. Exim therefore diagnoses a configuration error in this case. See **sender_host_accept_relay** for more details.

remote_max_parallel

Type: *integer*
Default: *1*

This option controls parallel delivery to remote sites. If the value is less than 2, parallel delivery is disabled, and Exim does all the remote deliveries for a message one by one, from a single delivery process. Otherwise, if a message has to be delivered to more than one remote host, or if several copies have to be sent to the same remote host, then up to **remote_max_parallel** deliveries are done simultaneously, each in a separate process. If more than **remote_max_parallel** deliveries are required, then the maximum number of processes are started, and as each one finishes, another is begun. The order of starting processes is the same as if sequential delivery were being done, and can be controlled by the **remote_sort** option. If parallel delivery takes place while running with debugging turned on, the debugging output from each delivery process is tagged with its process id.

The overhead in doing this is a fork to set up a separate process for each delivery, and the associated management of the subprocess (including getting back the result of the delivery attempt). As well as the process overhead, there may be a small additional penalty paid for parallel delivery. If a host is found to be down, this fact cannot be communicated to any deliveries that are running in parallel, though it will be passed on to any that start afterwards. This is no worse than if there were two separate messages being delivered simultaneously.

The option controls only the maximum number of parallel deliveries from one Exim process. Since Exim has no central queue manager, there is no way of controlling the total number of simultaneous deliveries if the configuration allows a delivery attempt as soon as a message is received. If you want to control the total number of deliveries on the system, then you need to set the **queue_only** option, which ensures that all incoming messages are simply added to the queue. Then set up an Exim daemon to start queue runner processes at appropriate intervals (probably fairly often, for example, every minute), and limit the total number of queue runners by setting the **queue_run_max** parameter. As each queue runner delivers only one message at a time, the maximum number of deliveries that can then take place at once is **queue_run_max** multiplied by **remote_max_parallel**.

If it is purely remote deliveries you want to control, then use **queue_smtp** instead of **queue_only**. This has the added benefit of doing the SMTP routing before queuing, so that several messages for the same host will eventually get delivered down the same connection.

remote_sort

Type: *domain-list*
Default: *unset*

When there are a number of remote deliveries for a message, they are sorted by domain into the order given by this list. For example,

```
remote_sort = "*.cam.ac.uk:*.uk"
```

would attempt to deliver to all addresses in the **cam.ac.uk** domain first, then to those in the **uk** domain, then to any others.

retry_interval_max

Type:	*time*
Default:	*24h*

Chapter 31 describes Exim's mechanisms for controlling the intervals between delivery attempts for messages that cannot be delivered straight away. This option sets an overall limit to the length of time between retries.

return_path_remove

Type:	*boolean*
Default:	*true*

RFC 822 states that the **Return-path:** header is 'added by the final transport system that delivers the message to its recipient' (section 4.3.1), which implies that this header should not be present in incoming messages. If this option is true, **Return-path:** headers are removed from messages as they are read. Exim's transports have options for adding **Return-path:** headers at the time of delivery. They are normally used only for final local deliveries.

return_size_limit

Type:	*integer*
Default:	*100K*

This option sets a limit in bytes on the size of messages that are returned to senders. If it is set to zero there is no limit. If the body of any message that is to be included in an error report is greater than the limit, it is truncated, and a comment pointing this out is added at the top. The actual cutoff may be greater than the value given, owing to the use of buffering for transferring the message in chunks. The idea is just to save bandwidth on those undeliverable 15-megabyte messages. If either the global or generic transport **message_size_limit** is set, the value of **return_size_limit** should be somewhat smaller.

rfc1413_hosts

Type:	*host-list*
Default:	*

RFC 1413 identification calls are made to any host which matches an item in the list. The items in the host list should not themselves contain ident data.

rfc1413_query_timeout

Type:	*time*
Default:	*30s*

This sets the timeout on RFC 1413 identification calls. If it is set to zero, no RFC 1413 calls are ever made.

security

Type:	*string*
Default:	*see below*

When **exim_user** is set non-zero in the runtime configuration or an Exim uid is compiled into the binary, Exim gives up root privilege for some of the time. As there are trade-offs between increased security and efficiency, this option is provided to control exactly how this is done. The option can be set to one of the strings 'seteuid', 'setuid', or 'setuid+seteuid', provided that a uid for Exim is defined. Otherwise it must be left unset. A full description of what these values mean is given in chapter 49. The default for this option is unset if no special Exim uid is defined, otherwise it is either 'setuid+seteuid' or 'setuid', depending on whether the **seteuid()** function is configured as being available or not.

sender_address_relay

Type: *address-list*
Default: *unset*

This option specifies a set of address patterns, one of which the sender of a message must match in order for the message to be accepted for relaying. If it is not set, all sender addresses are permitted. By default, this check operates in addition to any relaying checks on the sending host (see **host_accept_relay** above). However, if **relay_match_host_or_sender** is set, then either a host match or a sender match is sufficient to allow the relaying to proceed. For this reason, **sender_address_relay** is required to be set if **relay_match_host_or_sender** is set.

The rewrite flag X (see section 32.8) provides a special-purpose facility we have a use for in Cambridge. It adds additional checking to **sender_address_relay**. Whenever a sender address passes the check, if there are any rewriting rules with the X flag set, the address is rewritten using those rules, and if this makes any change to the address, the new address must verify successfully for the relaying to be permitted.

sender_reject

Type: *address-list*
Default: *unset*

This option can be set in order to reject mail from certain senders. The check is done on the sender's address as given in the MAIL command in SMTP, but not for local senders where the logged-in user's address is going to override anyway.

If the sender's address is source-routed, it is the final component of the address that is checked. The check is not done for batch SMTP input. If the check fails, a 550 return code is given to MAIL. This doesn't always stop remote mailers from trying again. See **sender_reject_recipients** for an alternative. Typical examples of the use of this option might be:

```
sender_reject = "spamuser@some.domain:spam.domain"
sender_reject = partial-dbm;/etc/mail/blocked/senders
```

Note that this check operates on sender address domains independently of the sending host; **host_reject** can be used to block all mail from particular hosts, while **host_accept_relay**, and **sender_address_relay** can be used to prevent unwanted relaying.

sender_reject_recipients

Type: *address-list*
Default: *unset*

This operates in exactly the same way as **sender_reject** except that the rejection is given in the form of a 550 error code to every RCPT command instead of rejecting MAIL. This seems to be the only way of saying 'no' to some mailers. Note that this is not an option for rejecting specific recipients. The way to do that is to set **receiver_verify** and arrange for those recipients to fail verification.

sender_try_verify

Type: *boolean*
Default: *false*

See **sender_verify**.

sender_unqualified_hosts

Type: *host-list*
Default: *unset*

This option lists those hosts from which Exim is prepared to accept unqualified sender addresses. The addresses are made fully qualified by the addition of **qualify_domain**. Typically the hosts are local ones, but if you want to imitate the behaviour of mailers that accept unqualified addresses from anywhere, specify

```
  sender_unqualified_hosts = *
```

sender_verify

Type:	*boolean*
Default:	*false*

If this option is true, envelope sender addresses on incoming SMTP messages are checked to ensure that they are valid. Messages with invalid envelope senders are rejected with a permanent error code if **sender_verify_reject** is set (the default). Otherwise a warning is logged. See section 39.2 for details of the rejection, which can happen at three different points in the SMTP dialogue. If a sender cannot immediately be verified, a temporary error code is returned after reading the data (so the headers can be logged). The **sender_try_verify** option is less severe: it operates in exactly the same way as **sender_verify** except that if an address cannot immediately be verified, it is accepted instead of being temporarily rejected.

sender_verify_batch

Type:	*boolean*
Default:	*false*

If this option is unset, then the **sender_verify** options are not applied to batched SMTP input.

sender_verify_hosts

Type:	*host-list*
Default:	*

If **sender_verify** or **sender_try_verify** is true, this option specifies a list of hosts and RFC 1413 identifications to which sender verification applies. The check caused by **headers_sender_verify** also happens only for matching hosts. See chapter 39 for further details.

sender_verify_fixup

Type:	*boolean*
Default:	*false*

Experience shows that many messages are sent out onto the Internet with invalid sender addresses in the envelopes (that is, in the MAIL command of the SMTP dialogue), but with valid addresses in the **Sender:**, **From:**, or **Reply-to:** header fields. If **sender_verify** and **sender_verify_reject** are true and this option is also true, an invalid envelope sender or one that cannot immediately be verified is replaced by a valid value from the headers. If **sender_verify_reject** is false, the envelope sender is not changed, but Exim writes a log entry giving the correction it would have made. See chapter 39 for details.

sender_verify_max_retry_rate

Type:	*integer*
Default:	*12*

If this option is greater than zero, and the rate of temporary rejection of a specific incoming sender address from a specific host, in units of rejections per hour, exceeds it, the temporary error is converted into a permanent verification error. Temporary rejections most commonly occur when a sender address cannot be verified because a DNS lookup fails to complete.

The intent of this option is to stop hosts hammering too frequently with temporarily failing sender addresses. The default value of 12 means that a sender address that has a temporary verification error more than once every 5 minutes will eventually get permanently rejected. Once permanent rejection has been triggered, subsequent temporary failures all cause permanent errors, until there has been an interval of at least 24 hours since the last failure. After 24 hours, the hint expires.

sender_verify_reject

Type:	*boolean*
Default:	*true*

When this is set, a message is rejected if sender verification fails. If it is not set, a warning message is written to the main and reject logs, and the message is accepted (unless some other error occurs).

smtp_accept_keepalive

Type:	*boolean*
Default:	*true*

This option controls the setting of the SO_KEEPALIVE option on incoming TCP/IP socket connections. This causes the kernel periodically to send some OOB (out-of-band) data on idle connections. The reason for doing this is that it has the beneficial effect of freeing up certain types of connection that can get stuck when the remote host is disconnected without tidying up the TCP/IP call properly.

smtp_accept_max

Type:	*integer*
Default:	*20*

This specifies the maximum number of simultaneous incoming SMTP calls that Exim will accept. It applies only to the listening daemon; there is no control (in Exim) when incoming SMTP is being handled by **inetd**. If the value is set to zero, no limit is applied. However, it is required to be non-zero if **smtp_accept_max_per_host** or **smtp_accept_queue** is set.

smtp_accept_max_per_host

Type:	*integer*
Default:	*0*

This option restricts the number of simultaneous IP connections from a single host (strictly, from a single IP address) to the Exim daemon. The default value of zero imposes no limit. If this option is set, it is required that **smtp_accept_max** be set non-zero.

smtp_accept_queue

Type:	*integer*
Default:	*0*

If the number of simultaneous incoming SMTP calls handled via the listening daemon exceeds this value, then messages received are simply placed on the queue, and no delivery processes are started automatically. A value of zero implies no limit, and clearly any non-zero value is useful only if it is less than the **smtp_accept_max** value (unless that is zero). See also **queue_only**, **queue_only_load**, **queue_smtp**, and the various **-od** command line options.

smtp_accept_queue_per_connection

Type:	*integer*
Default:	*10*

This option limits the number of delivery processes that Exim starts automatically when receiving messages via SMTP, whether via the daemon or by the use of **-bs** or **-bS**. If the value of the option is greater than zero, and the number of messages received in a single SMTP session exceeds this number, subsequent messages are placed on the spool, but no delivery process is started. This helps to limit the number of Exim processes when a server restarts after downtime and there is a lot of mail waiting for it on other systems. On large systems the default should probably be increased.

smtp_accept_reserve

Type: *integer*
Default: *0*

When **smtp_accept_max** is set greater than zero, this option specifies a number of SMTP connections that are reserved for connections from the hosts that are specified in **smtp_reserve_hosts**. The value set in **smtp_accept_max** includes this reserve pool. For example, if **smtp_accept_max** is set to 50 and **smtp_accept_reserve** is set to 5, then once there are 45 active connections, new ones are accepted only from hosts listed in **smtp_reserve_hosts**.

smtp_banner

Type: *string*
Default: *see below*

This string, which is expanded every time it is used, is output as the initial positive response to an SMTP connection. The default setting is:

```
smtp_banner = "${primary_hostname} ESMTP Exim ${version_number} \
  #${compile_number} ${tod_full}"
```

Failure to expand the string causes a panic error. If you want to create a multiline response to the initial SMTP connection, use '\n' in the string at appropriate points, but not at the end. Note that the 220 code is not included in this string. Exim adds it automatically (several times in the case of a multiline response).

smtp_check_spool_space

Type: *boolean*
Default: *true*

When this option is set, if an incoming SMTP session encounters the SIZE option on a MAIL command, it checks that there is enough space in the spool directory's partition to accept a message of that size, while still leaving free the amount specified by **check_spool_space** (even if that value is zero). If there isn't enough space, a temporary error code is returned.

smtp_connect_backlog

Type: *integer*
Default: *5*

This specifies a maximum number of waiting SMTP connections. Exim passes this value to the TCP/IP system when it sets up its listener. Once this number of connections are waiting for the daemon's attention, subsequent connection attempts are refused at the TCP/IP level. At least, that is what the manuals say. In Solaris 2.4 such connection attempts have been observed to time out. The default value of 5 is a conservative one, suitable for older and smaller systems. For large systems is it probably a good idea to increase this, possibly substantially (to 50, say). It also gives some protection against denial-of-service attacks by SYN flooding.

smtp_etrn_command

Type: *string*
Default: *unset*

If this option is set, the given command is run whenever an SMTP ETRN command is received from a host that is permitted to issue such commands (see **smtp_etrn_hosts** below). The string is split up into separate arguments which are independently expanded. The expansion variable **$domain** is set to the argument of the ETRN command, and no syntax checking is done on it. For example:

```
smtp_etrn_command = /etc/etrn_command $domain $sender_host_address
```

A new process is created to run the command, and Exim does not wait for it to complete. Consequently, its status cannot be checked. As Exim is normally running under its own uid when receiving SMTP, it is not possible for it to change the uid before running the command.

You must disable **smtp_etrn_serialize** if you use this option to run something other than a call of Exim with the **-R** option, because otherwise the serialization lock never gets removed.

smtp_etrn_hosts

Type:	*host-list*
Default:	*unset*

RFC 1985 describes an SMTP command called ETRN which is designed to overcome the security problems of the TURN command (which has fallen into disuse). Exim recognizes ETRN if the calling host matches an entry in **smtp_etrn_hosts**. The ETRN command is concerned with 'releasing' messages that are awaiting delivery to certain hosts. As Exim does not organize its message queue by host, the only form of ETRN that is supported by default is the one where the text starts with the '#' prefix, where the remainder of the text is specific to the SMTP server.

However, if the **smtp_etrn_command** option is set, the given command is run for every ETRN command, whatever the form of the argument. In the absence of **smtp_etrn_command**, a valid ETRN command causes a run of Exim with the **-R** option to happen, with the remainder of the ETRN text (following the '#') as its argument. For example,

```
  ETRN #brigadoon
```

causes a delivery attempt on all messages with undelivered addresses containing the text 'brigadoon'. Because a separate delivery process is run to do the delivery, there is no security risk with ETRN.

smtp_etrn_serialize

Type:	*boolean*
Default:	*true*

When this option is set, it prevents the simultaneous execution of more than one queue run for the same argument string as a result of an ETRN command. Exim implements serialization by means of a hints database in which a record is written whenever a process is started by ETRN, and deleted when a **-R** queue run completes. If you use **smtp_etrn_command** to do something other than run Exim with the **-R** option, you must disable **smtp_etrn_serialize** because otherwise the hints will never get deleted.

Obviously there is scope for hints records to get left lying around if there is a system or program crash. To guard against this, Exim ignores any records that are more than six hours old, but you should normally arrange to delete any files in the **spool/db** directory whose names begin with **serialize-** after a reboot.

smtp_expn_hosts

Type:	*host-list*
Default:	*unset*

The SMTP EXPN command is supported only if the calling host matches **smtp_expn_hosts**. You must add 'localhost' explicitly if you want calls to 127.0.0.1 to be able to use it. A single-level expansion of the address is done, as if the address were being tested using the **-bt** option. If an unqualified local part is given, it is qualified with **qualify_domain**. There is a generic option for directors which permits them to be skipped when processing an EXPN command (compare with verification).

smtp_load_reserve

Type:	*fixed-point*
Default:	*unset*

If the system load average ever gets higher than this, incoming SMTP calls are accepted only from those hosts that match an entry in **smtp_reserve_hosts**. There are some operating systems for which Exim cannot determine the load average (see chapter 1); for these this option has no effect.

smtp_receive_timeout

Type: *time*
Default: *5m*

This sets a timeout value for SMTP reception. If a line of input (either an SMTP command or a data line) is not received within this time, the SMTP connection is dropped and the message is abandoned. For non-SMTP input, the reception timeout is controlled by **accept_timeout**.

smtp_reserve_hosts

Type: *host-list*
Default: *unset*

This option defines hosts for which SMTP connections are reserved; see **smtp_accept_reserve** and **smtp_load_reserve** above.

smtp_verify

Type: *boolean*
Default: *false*

If this option is true, the SMTP command VRFY is supported on incoming SMTP connections; otherwise it is not.

split_spool_directory

Type: *boolean*
Default: *false*

If this option is set, it causes Exim to split its input directory into 62 subdirectories, each with a single alphanumeric character as its name. The fifth character of the message id is used to allocate messages to subdirectories; this is the least significant base-62 digit of the time of arrival of the message.

Splitting up the spool in this way may provide better performance on systems where there are long mail queues, by reducing the number of files in any one directory. The msglog directory is also split up in a similar way to the input directory; however, if **preserve_message_logs** is set, all old msglog files are still placed in the single directory **msglog.OLD**.

It is not necessary to take any special action for existing messages when changing **split_spool_directory**. Exim notices messages that are in the 'wrong' place, and continues to process them. If the option is turned off after a period of being on, the subdirectories will eventually empty and get deleted.

spool_directory

Type: *string*
Default: *compile-time configured (may be unset)*

This defines the directory in which Exim keeps its mail spool. The default value is taken from the compile-time configuration setting, if there is one. If not, this option must be set. The string is expanded, so it can contain, for example, a reference to **${primary_hostname}**.

If the spool directory name is fixed on your installation, it is recommended that you set it at build time rather than from this option, particularly if the log files are being written to the spool directory (see **log_file_path**). Otherwise log files cannot be used for errors that are detected early on, such as failures in the configuration file.

Even with a compiled-in path, however, this option makes it possible to run testing configurations of Exim without using the standard spool.

strip_excess_angle_brackets

Type:	*boolean*
Default:	*false*

If this option is set, then redundant pairs of angle brackets round ‘route-addr’ items in addresses are stripped. For example, **<<xxx@a.b.c.d>>** is treated as **<xxx@a.b.c.d>**. If this is in the envelope and the message is passed on to another MTA, the excess angle brackets are not passed on. If this option is not set, multiple pairs of angle brackets cause a syntax error.

strip_trailing_dot

Type:	*boolean*
Default:	*false*

If this option is set, a trailing dot at the end of a domain in an address is ignored. If this is in the envelope and the message is passed on to another MTA, the dot is not passed on. If this option is not set, a dot at the end of a domain causes a syntax error.

trusted_groups

Type:	*string-list*
Default:	*unset*

If this option is set, then any process that is running in one of the listed groups may pass a message to Exim and specify the sender’s address using the **-f** command line option, without Exim’s adding a **Sender:** header. If neither **trusted_groups** nor **trusted_users** is set, then only root and the Exim user can do this.

trusted_users

Type:	*string-list*
Default:	*unset*

If this option is set, then any process that is running as one of the listed users may pass a message to Exim and specify the sender’s address using the **-f** command line option, without Exim’s adding a **Sender:** header. If neither **trusted_users** nor **trusted_groups** is set, then only root and the Exim user can do this.

unknown_login

Type:	*string*
Default:	*unset*

This is a specialized feature for use in unusual configurations. By default, if the uid of the caller of Exim cannot be looked up using **getpwuid()**, Exim gives up. The **unknown_login** option can be used to set a login name to be used in this circumstance. It is expanded, so values like **user$caller_uid** can be set. When **unknown_login** is used, the value of **unknown_username** is used for the user’s real name (gecos field), unless this has been set by the **-F** option.

unknown_username

Type:	*string*
Default:	*unset*

See **unknown_login**.

uucp_from_pattern

Type:	*string*
Default:	*see below*

Some applications that pass messages to an MTA via a command line interface use an initial line starting with ‘From’ to pass the envelope sender. In particular, this is used by UUCP software. Exim recognizes such a line by means of a regular expression that is set in **uucp_from_pattern**, and when the pattern matches, the sender address is constructed by expanding the contents of

uucp_from_sender, provided that the caller of Exim is a trusted user. The default pattern recognizes lines in the following two forms:

```
From ph10 Fri Jan  5 12:35 GMT 1996
From ph10 Fri, 7 Jan 97 14:00:00 GMT
```

The pattern can be seen by running 'exim -bP uucp_from_pattern'. It checks only up to the hours and minutes, and allows for a 2-digit or 4-digit year in the second case. The first word after 'From' is matched in the regular expression by a parenthesized subpattern. The default value for **uucp_from_sender** is '$1', which therefore just uses this first word ('ph10' in the example above) as the message's sender. See also **ignore_fromline_hosts**.

uucp_from_sender

Type: *string*
Default: *"$1"*

See **uucp_from_pattern** above.

warnmsg_file

Type: *string*
Default: *unset*

This option defines a template file containing paragraphs of text to be used for constructing the warning message which is sent by Exim when a message has been on the queue for a specified amount of time, as specified by **delay_warning**. Details of the file's contents are given in chapter 33. See also **errmsg_file**.

12. Driver specifications

The second, third, and fourth parts of Exim's configuration file specify which transport, director, and router drivers are to be used. Directors and routers are similar, in that an address is passed to a list of them in the order in which they are defined, whereas the order in which transports are specified is immaterial, because a transport is invoked only after being passed an address by a director or a router. Section 3.4 discusses how the different kinds of driver interact.

The format of the configuration data is the same for all three types of driver, and is as follows:

```
<driver instance name>:
  <option>
  ...
  <option>
```

There are two kinds of option: *generic* and *private*. The generic options are those that apply to all drivers of the same type (that is, all directors, or all routers, or all transports). There is always at least one generic option, called **driver**, which specifies which particular driver is being used. The private options are particular to each driver, and none need appear.

The options may appear in any order, except that the **driver** option must precede any private options, since these depend on the particular driver. For this reason, it is recommended that **driver** always be the first option.

In earlier versions of Exim, commas were used between options, and the generic options had to precede the private ones and be terminated by a semicolon. This has not been the case for some time, and at release 3.00 the backwards-compatibility code for ignoring commas and semicolons was removed.

Each instance of a driver is given an identifying name for reference in logging and elsewhere. The name can be any sequence of letters, digits, and underscores (starting with a letter) and must be unique among drivers of the same type. A router and a transport (for example) can each have the same name, but no two router instances can have the same name. The name of a driver instance should not be confused with the name of the underlying driver. The configuration lines

```
remote_smtp:
  driver = smtp
```

create an instance of the **smtp** transport driver whose name is **remote_smtp**. The same driver code can be used more than once, with different instance names and different option settings each time. A second instance of the **smtp** transport, with different options, might be defined thus:

```
special_smtp:
  driver = smtp
  service = 1234
  command_timeout = 10s
```

The names **remote_smtp** and **special_smtp** would be used to reference these driver instances from directors or routers, and would appear in log lines.

Comment lines may appear in the middle of driver specifications. The full list of option settings for any particular driver instance, including all the defaults, can be extracted by making use of the **-bP** command line option (see chapter 5).

The next chapter describes the environment in which local deliveries are done, and how this is affected by the configurations of the relevant directors, routers, and transports. Then there is a chapter describing the generic options for transports, followed by descriptions of the available transport drivers. Directors and routers have some generic options in common, and these are covered in chapter 19 before the descriptions of the generic options that are specific to each type of driver, and the drivers themselves.

[95] *driver specifications (12)*

13. Environment for running local transports

Local transports handle deliveries to files and pipes. (The **autoreply** transport can be thought of as similar to a pipe.) Whenever a local transport is run, Exim forks a subprocess for it. Before running the transport code, it sets a specific uid and gid by calling **setuid()** and **setgid()**. It also sets a current file directory; for some transports a home directory setting is also relevant. The **pipe** transport is the only one which sets up environment variables; see section 17.3 for details.

The values used for the uid, gid, and the directories may come from several different places. In many cases the director that handles the address associates settings with that address. However, values may also be given in the transport's own configuration, and these override anything that comes with the address. The sections below contain a summary of the possible sources of the values, and how they interact with each other.

13.1 Uids and gids

All local transports have the options **group** and **user**. If **group** is set, it overrides any group that may be set in the address, even if **user** is not set. This makes it possible, for example, to run local mail delivery under the uid of the recipient, but in a special group. For example:

```
group_delivery:
  driver = appendfile
  file = /var/spool/mail/${local_part}
  group = mail
```

If **user** is set for a transport, its value overrides what is set in the address. If **user** is non-numeric and **group** is not set, the gid associated with the user is used. If **user** is numeric, then **group** must be set.

The **pipe** transport contains the special option **pipe_as_creator**. If this is set and **user** is not set, the uid of the process that called Exim to receive the message is used, and if **group** is not set, the corresponding original gid is also used.

When the uid is taken from the transport's configuration, the **initgroups()** function is called for the groups associated with that uid if the **initgroups** option is set for the transport; **pipe** is the only transport that has such an option.

When the uid is not specified by the transport, but is associated with the address by a director or router, the option for calling **initgroups()** is taken from the director or router configuration. All directors and routers have **group**, **user**, and **initgroups** options, which are used as follows:

For the **aliasfile** director they specify the uid and gid for local deliveries generated directly – that is, deliveries to pipes or files. They have no effect on generated addresses that are processed independently.

The **forwardfile** director's **check_local_user** option causes a password file lookup for the local part of an address. The uid and gid obtained from this lookup are used for any directly generated local deliveries, but they can be overridden by the **group** and **user** options of the director. As for **aliasfile**, these values are not used for generated addresses that are processed independently.

The **localuser** director looks up local parts in the password file, and sets the uid and gid from that file for local deliveries, but these values can be overridden by the director's options.

For the **smartuser** director and all the routers, the **group**, **user**, and **initgroups** options are used only if the driver sets up a delivery to a local transport.

13.2 Current and home directories

The **pipe** transport has a **home_directory** option. If this is set, it overrides any home directory set by the director for the address. The value of the home directory is set in the environment variable HOME while running the pipe. It need not be set, in which case HOME is not defined.

The **appendfile** transport does not have a **home_directory** option. The only use for a home directory in this transport is if the expansion variable **$home** is used in one of its options, in which case the value set by the director is used.

The **appendfile** and **pipe** transports have a **current_directory** option. If this is set, it overrides any current directory set by the director for the address. If neither the director nor the transport sets a current directory, then Exim uses the value of the home directory, if set. Otherwise it sets the current directory to '/' before running a local transport.

The **aliasfile**, **forwardfile**, and **localuser** directors all have **current_directory** and **home_directory** options, which are associated with any addresses they explicitly direct to a local transport.

For **forwardfile**, if **home_directory** is not set and there is a **file_directory** value, that is used instead. If it too is not set, but **check_local_user** is set, the user's home directory is used. For **localuser**, if **home_directory** is not set, the home directory is taken from the password file entry that this director looks up. There are no defaults for **current_directory** in the directors, because it defaults to the value of **home_directory** if it is not set at transport time.

The **smartuser** director and all the routers have no means of setting up home and current directory strings; consequently any local transport that they use must specify them for itself if they are required.

13.3 Expansion variables derived from the address

Normally a local delivery is handling a single address, and in that case the variables such as **$domain** and **$local_part** are set during local deliveries. However, in some circumstances more than one address may be handled at once (for example, while writing batch SMTP for onward transmission by some other means). In this case, the variables associated with the local part are never set, **$domain** is set only if all the addresses have the same domain, and **$original_domain** is never set.

14. Generic options for transports

The generic options for transports are as follows:

body_only

Type: *boolean*
Default: *false*

If this option is set, the message's headers are not transported. It is mutually exclusive with **headers_only**. If it is used with the **appendfile** or **pipe** transports, the settings of **prefix** and **suffix** should be checked, since this option does not automatically suppress them.

debug_print

Type: *string*
Default: *unset*

If this option is set and debugging is enabled (see **-d**, **-v**, and **debug_level**), then the string is expanded and included in the debugging output when the transport is run. This is to help with checking out the values of variables and so on when debugging driver configurations. For example, if a **headers_add** option is not working properly, **debug_print** could be used to output the variables it references. A newline is added to the text if it does not end with one.

delivery_date_add

Type: *boolean*
Default: *false*

If this option is true, a **Delivery-date:** header is added to the message. This gives the actual time the delivery was made. As this is not a standard header, Exim has a configuration option (**delivery_date_remove**) which requests its removal from incoming messages, so that delivered messages can safely be resent to other recipients.

driver

Type: *string*
Default: *unset*

This specifies which of the available transport drivers is to be used. For example:

```
  driver = smtp
```

There is no default, and this option must be set for every transport.

envelope_to_add

Type: *boolean*
Default: *false*

If this option is true, an **Envelope-to:** header is added to the message. This gives the original address(es) in the incoming envelope that caused this delivery to happen. More than one address may be present if **batch** or **bsmtp** is set on transports that support them, or if more than one original address was aliased or forwarded to the same final address. As this is not a standard header, Exim has a configuration option (**envelope_to_remove**) which requests its removal from incoming messages, so that delivered messages can safely be resent to other recipients.

headers_add

Type: *string*
Default: *unset*

This option specifies a string of text which is expanded and added to the header portion of a message as it is transported. If the result of the expansion is an empty string, or if the expansion is forced to fail, no action is taken. Other expansion failures are treated as errors and cause the delivery to be deferred. The expanded string should be in the form of one or more RFC 822 header lines, separated by newlines (coded as '\n' inside a quoted string), for example:

```
headers_add = "X-added: this is a header added at $tod_log\n\
               X-added: this is another"
```

Exim does not check the syntax of these added headers. A newline is supplied at the end if one is not present. The text is added at the end of any existing headers. If you include a blank line within the string, you can subvert this facility into adding text at the start of the message's body.

The name **add_headers** was formerly used for this option, and is retained as a synonym for backward compatibility. Additional headers can also be specified by directors and routers. See chapter 19 and section 43.13.

headers_only

Type:	*boolean*
Default:	*false*

If this option is set, the message's body is not transported. It is mutually exclusive with **body_only**. If it is used with the **appendfile** or **pipe** transports, the settings of **prefix** and **suffix** should be checked, since this option does not automatically suppress them.

headers_remove

Type:	*string*
Default:	*unset*

This option consists of a colon-separated list of header names, not including the terminating colon, for example:

```
headers_remove = "return-recipt-to:acknowledge-to"
```

Any existing headers matching those names are not included in any message that transmitted by the transport. However, added headers may have these names. Thus it is possible to replace a header by specifying it in **remove_headers** and supplying the replacement in **add_headers**.

The name **remove_headers** was formerly used for this option, and is retained as a synonym for backward compatibility. Headers to be removed can also be specified by directors and routers. See chapter 19 and section 43.13.

message_size_limit

Type:	*integer*
Default:	*0*

This option controls the size of messages passed through the transport. If its value is greater than zero and the size of a message message exceeds the limit, the address is failed. If there is any chance that the resulting bounce message could be routed to the same transport, you should ensure that **return_size_limit** is less than the transport's **message_size_limit**, as otherwise the bounce message will fail to get delivered.

return_path

Type:	*string*
Default:	*unset*

If this option is set, the string is expanded at transport time and replaces the existing return path (envelope sender) value. The expansion can refer to the existing value via **$return_path**. If the expansion is forced to fail, no replacement occurs; if it fails for another reason, Exim panics. This option can be used to support VERP (Variable Envelope Return Paths) – see chapter 42.

return_path_add

Type: *boolean*
Default: *false*

If this option is true, a **Return-path:** header is added to the message. Although the return path is normally available in the prefix line of BSD mailboxes, this is commonly not displayed by MUAs, and so the user does not have easy access to it.

RFC 822 states that the **Return-path:** header is 'added by the final transport system that delivers the message to its recipient' (section 4.3.1), which implies that this header should not be present in incoming messages. Exim has a configuration option, **return_path_remove**, which requests removal of this header from incoming messages, so that delivered messages can safely be resent to other recipients.

shadow_condition

Type: *string*
Default: *unset*

See **shadow_transport** below.

shadow_transport

Type: *string*
Default: *unset*

This facility is somewhat experimental, and may change in future. A local transport may set the **shadow_transport** option to the name of another, previously-defined, local transport. Shadow remote transports are not supported.

Whenever a delivery to the main transport succeeds, and either **shadow_condition** is unset, or its expansion does not result in a forced expansion failure or the empty string or one of the strings '0' or 'no' or 'false', the message is also passed to the shadow transport. However, the result of the shadow transport is discarded and does not affect the subsequent processing of the message. Only a single level of shadowing is provided; the **shadow_transport** option is ignored on any transport when it is running as a shadow. Options concerned with output from pipes are also ignored.

The log line for the successful delivery has an item added on the end, of the form

`ST=`<*shadow transport name*>

If the shadow transport did not succeed, the error message is put in parentheses afterwards.

Shadow transports can be used for a number of different purposes, including keeping more detailed log information than Exim normally provides, and implementing automatic acknowledgement policies based on message headers that some sites insist on.

transport_filter

Type: *string*
Default: *unset*

This option sets up a filtering (in the Unix shell sense) process for messages at transport time. It should not be confused with mail filtering as set up by individual users.

When the message is about to be written out, the command specified by **transport_filter** is started up in a separate process, and the entire message, including the headers, is passed to it on its standard input (this in fact is done from a third process, to avoid deadlock). This happens before any SMTP-specific processing, such as turning '\n' into '\r\n' and escaping lines beginning with a dot.

The filter's standard output is read and written to the message's destination. The filter can perform any transformations it likes, but of course should take care not to break RFC 822 syntax. A demonstration Perl script is provided in **util/transport-filter.pl**; this makes a few arbitrary modifications just to show the possibilities.

A problem might arise if the filter increases the size of a message that is being sent down an SMTP channel. If the receiving SMTP server has indicated support for the SIZE parameter, Exim will have sent the size of the message at the start of the SMTP session. If what is actually sent is substantially more, the server might reject the message. This can be worked round by setting the **size_addition** option on the **smtp** transport, either to allow for additions to the message, or to disable the use of SIZE altogether.

The value of the option is the command string for starting up the filter, which is run directly from Exim, not under a shell. The string is parsed by Exim in the same way as a command string for the **pipe** transport: Exim breaks it up into arguments and then expands each argument separately. The special argument **$pipe_addresses** is replaced by a number of arguments, one for each address that applies to this delivery. (This isn't an ideal name for this feature here, but as it was already implemented for the **pipe** transport, it seemed sensible not to change it.)

The expansion variables **$host** and **$host_address** are available when the transport is a remote one. They are set only for the expansion of a transport filter command, as that is the only thing that is expanded after a connection has been set up. For example:

```
transport_filter = "/some/directory/transport-filter.pl \
  $host $host_address $sender_address $pipe_addresses"
```

The filter process is run under the same uid and gid as the normal delivery. For remote deliveries this is the exim uid/gid if they are defined.

If a transport filter is set on an autoreply transport, the original message is passed through the filter as it is being copied into the newly generated message, which happens if the **return_message** option is set.

15. The appendfile transport

The **appendfile** transport delivers a message by appending it to a file in the local file system, or by creating an entirely new file in a specified directory. Single files to which messages are appended can be in the traditional Unix mailbox format, or optionally in the MBX format supported by the Pine MUA and University of Washington IMAP daemon, *inter alia*. When each message is being delivered as a separate file, 'maildir' format can optionally be used to give added protection against failures that happen part-way through the delivery. A third form of separate-file delivery known as 'mailstore' is also supported. For all file formats, Exim attempts to create as many levels of directory as necessary, provided that **create_directory** is set.

The code for the optional formats is not included in the Exim binary by default. It is necessary to set SUPPORT_MBX, SUPPORT_MAILDIR and/or SUPPORT_MAILSTORE in **Local/Makefile** to have the appropriate code included.

Appendfile can be used by routers as a pseudo-remote transport for putting messages into files for remote delivery by some means other than Exim, though it is more commonly used by directors for local deliveries to users' mailboxes. It is also used for delivering messages to files or directories whose names are obtained directly from alias, forwarding, or filtering operations. In these cases, **$local_part** contains the local part that was aliased or forwarded, while **$address_file** contains the name of the file or directory.

As **appendfile** is a local transport, it is always run in a separate process, under a non-privileged uid and gid, which are set by **setuid()**. In the common local delivery case, these are the uid and gid belonging to the user to whom the mail is being delivered. The current directory is also normally set to the user's home directory. See chapter 13 for a discussion of the local delivery environment.

If the transport fails for any reason, the message remains on the input queue so that there can be another delivery attempt later. If there is an error while appending to a file (for example, quota exceeded or partition filled), Exim attempts to reset the file's length and last modification time back to what they were before. Exim supports a local quota, for use when the system facility is unavailable or cannot be used for some reason.

Before appending to a file, a number of security checks are made, and the file is locked. A detailed description is given below, after the list of private options.

15.1 Private options for appendfile

allow_symlink

Type:	*boolean*
Default:	*false*

By default, **appendfile** will not deliver if the path name for the file is that of a symbolic link. Setting this option relaxes that constraint, but there are security issues involved in the use of symbolic links. Be sure you know what you are doing if you set this. Details of exactly what this option affects are included in the discussion which follows this list of options.

batch

Type:	*string*
Default:	*"none"*

Normally, each address that is directed or routed to an **appendfile** transport is handled separately. In special cases it may be desirable to handle several addresses at once, for example, when passing a message with several addresses to a different mail regime (for example, UUCP), though this is more often done using the **pipe** transport. If this option is set to the string 'domain', then all addresses with the same domain that are directed or routed to the transport are handled in a single delivery. If it is set to 'all' then multiple domains are batched. The list of addresses is included in the **Envelope-to:** header if **envelope_to_add** is set (see below). The only difference between this option and **bsmtp** is the inclusion of SMTP command lines in the output for **bsmtp**.

batch_max

Type: *integer*
Default: *100*

This limits the number of addresses that can be handled in a batch, and applies to both the **batch** and the **bsmtp** options.

bsmtp

Type: *string*
Default: *"none"*

This option is used to set up an **appendfile** transport as a pseudo-remote transport for delivering messages into local files in batch SMTP format for onward transmission by some non-Exim means. It is usually necessary to suppress the default settings of the **prefix** and **suffix** options when using batch SMTP. The value of the option must be one of the strings ‘none’, ‘one’, ‘domain’, or ‘all’. The first of these turns the feature off. A full description of the batch SMTP mechanism is given in section 42.7. When **bstmp** is set, the **batch** option automatically takes the same value. See also the **use_crlf** option.

bsmtp_helo

Type: *boolean*
Default: *false*

When this option is set, a HELO line is added to the output at the start of each message written in batch SMTP format. Some software that reads batch SMTP is unhappy without this.

check_group

Type: *boolean*
Default: *false*

The group owner of the file is checked to see that it is the same as the group under which the delivery process is running when this option is set. The default setting is unset because the default file mode is 0600, which means that the group is irrelevant.

create_directory

Type: *boolean*
Default: *true*

When this option is true, Exim creates any missing superior directories for the file that it is about to write. A created directory’ s mode is given by the **directory_mode** option.

create_file

Type: *string*
Default: *"anywhere"*

This option constrains the location of files that are created by this transport. It must be set to one of the words ‘anywhere’, ‘inhome’, or ‘belowhome’. In the second and third cases, a home directory must have been set up for the address by the director that handled it. This option isn’t useful when an explicit file name is given for normal mailbox deliveries; it is intended for the case when file names have been generated from user’s **.forward** files, which are usually handled by an **appendfile** transport called **address_file**. See also **file_must_exist**.

current_directory

Type: *string*
Default: *unset*

If this option is set, it specifies the directory to make current when running the delivery process. The string is expanded at the time the transport is run. See chapter 13 for details of the local delivery environment.

directory

Type: *string*
Default: *unset*

This option is mutually exclusive with the **file** option. When it is set, the string is expanded, and the message is delivered into a new file or files in or below the given directory, instead of being appended to a single mailbox file. See section 15.3 for details of this form of delivery.

directory_mode

Type: *octal integer*
Default: *0700*

If **appendfile** creates any directories as a result of the **create_directory** option, their mode is specified by this option.

file

Type: *string*
Default: *unset*

This option need not be set when **appendfile** is being used to deliver to files whose names are obtained from forwarding, filtering, or aliasing address expansions (by default under the instance name **address_file**), as in those cases the file name is associated with the address. Otherwise, the **file** option must be set unless the **directory** option is set. Either **use_fcntl_lock** or **use_lockfile** (or both) must be set with **file**. If you are using more than one host to deliver over NFS into the same mailboxes, you should always use lock files.

The string value is expanded for each delivery, and must yield an absolute path. If the expansion contains a reference to the **local_part** variable, this is checked to ensure that it does not contain a forward slash character – to prevent an unexpected change of directory. The most common settings of this option are variations on one of these examples:

```
file = /var/spool/mail/${local_part}
file = /home/${local_part}/inbox
file = ${home}/inbox
```

In the first example, all deliveries are done into the same directory. If Exim is configured to use lock files (see **use_lockfile** below) it must be able to create a file in the directory, so the 'sticky' bit must be turned on for deliveries to be possible, or alternatively the **group** option can be used to run the delivery under a group id which has write access to the directory.

If there is no file name, or the expansion fails, or a local part contains a forward slash character, a delivery error occurs.

file_must_exist

Type: *boolean*
Default: *false*

If this option is true, the file specified by the **file** option must exist, and an error occurs if it does not. Otherwise, it is created if it does not exist.

from_hack

Type: *boolean*
Default: *true*

If this option is true, lines in the body of the message that start with the string 'From ' are modified by adding a right angle-bracket at their start. This is necessary for traditional BSD-format mailboxes, where such lines might otherwise indicate the start of a new message.

group

Type: *string*
Default: *unset*

If this option is set, it specifies the group under whose gid the delivery process is to be run, and, if **check_group** is set, the group owner of an existing file to which the message is to be appended. If the option is not set, a value associated with a user may be used (see below); otherwise a value must have been associated with the address by the director which handled it. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the transport is run, and must yield either a digit string or a name which can be looked up using **getgrnam()**.

The **group** option is commonly set for local deliveries on systems where the set of user mailboxes is in a single directory owned by a group such as 'mail'. Note that it should *not* be set on the instance of **appendfile** that is used for deliveries to files specified by users in their forward files (called **address_file** in the default configuration), because such deliveries should take place under the individual users' personal uids and gids.

lock_interval

Type: *time*
Default: *3s*

This specifies the time to wait between attempts to lock the file. See below for details of locking.

lock_retries

Type: *integer*
Default: *10*

This specifies the maximum number of attempts to lock the file. A value of zero is treated as 1. See below for details of locking.

lockfile_mode

Type: *octal integer*
Default: *0600*

This specifies the mode of the created lock file, when a lock file is being used (see **use_lockfile**).

lockfile_timeout

Type: *time*
Default: *30m*

When a lock file is being used (see **use_lockfile**), if a lock file already exists and is older than this value, it is assumed to have been left behind by accident, and Exim attempts to remove it.

maildir_format

Type: *boolean*
Default: *false*

If this option is set with the **directory** option, then the delivery is into a new file in the 'maildir' format that is used by some other mail software. The option is available only if SUPPORT_MAILDIR is present in **Local/Makefile**. See section 15.3 below for further details.

maildir_retries

Type: *integer*
Default: *10*

This option specifies the number of times to retry when writing a file in 'maildir' format. See section 15.3 below.

maildir_tag

Type: *string*
Default: *unset*

This option applies only to deliveries in maildir format, and is described in section 15.3 below.

mailstore_format

Type: *boolean*
Default: *false*

If this option is set with the **directory** option, then the delivery is into two new files in 'mailstore' format. The option is available only if SUPPORT_MAILSTORE is present in **Local/Makefile**. See section 15.3 below for further details.

mailstore_prefix

Type: *string*
Default: *unset*

This option applies only to deliveries in mailstore format, and is described in section 15.3 below.

mailstore_suffix

Type: *string*
Default: *unset*

This option applies only to deliveries in mailstore format, and is described in section 15.3 below.

mbx_format

Type: *boolean*
Default: *false*

This option is available only if Exim has been compiled with SUPPORT_MBX set in **Local/Makefile**. If **mbx_format** is set with the **file** option, then the message is appended to the mailbox file in MBX format instead of traditional Unix format. This format is supported by Pine4 and its associated IMAP and POP daemons, and is implemented by the **c-client** library that they all use. The **prefix** and **suffix** options are not automatically changed by the use of **mbx_format**; they should normally be set empty.

If none of the locking options are mentioned in the configuration, **use_mbx_lock** is assumed and the other locking options default to false. It is possible to specify the other kinds of locking with **mbx_format**, but **use_fcntl_lock** and **use_mbx_lock** are mutually exclusive. MBX locking interworks with **c-client**, providing for shared access to the mailbox. It should not be used if any program that does not use this form of locking is going to access the mailbox, nor should it be used if the mailbox file is NFS mounted, because it works only when the mailbox is accessed from a single host.

If you set **use_fcntl_lock** with an MBX-format mailbox, you cannot use the standard version of **c-client**, because as long as it has a mailbox open (this means for the whole of a Pine or IMAP session), Exim will not be able to append messages to it.

mode

Type: *octal integer*
Default: *0600*

If the output file is created, it is given this mode. If it already exists and has wider permissions, they are reduced to this mode. If it has narrower permissions, an error occurs unless **mode_fail_narrower** is false. However, if the delivery is the result of a **save** command in a filter file specifing a particular mode, then the mode of the output file is always forced to take that value, and this option is ignored.

mode_fail_narrower

Type: *boolean*
Default: *true*

This option applies in the case when an existing mailbox file has a narrower mode than that specified by the **mode** option. If **mode_fail_narrower** is true, the delivery is frozen ('mailbox has the wrong mode'); otherwise Exim continues with the delivery attempt, using the existing mode of the file.

notify_comsat

Type: *boolean*
Default: *false*

If this option is true, the **comsat** daemon is notified after every successful delivery to a user mailbox. This is the daemon that notifies logged on users about incoming mail.

prefix

Type: *string*
Default: *see below*

The string specified here is expanded and output at the start of every message. The default is

```
prefix = "From ${if def:return_path{$return_path}{MAILER-DAEMON}}\
  ${tod_bsdinbox}\n"
```

This line can be suppressed by setting

```
prefix =
```

and this is usually necessary when doing batch SMTP deliveries, or delivering into individual files or MBX-format mailboxes.

quota

Type: *string*
Default: *unset*

This option imposes a limit on the size of the file to which Exim is appending, or to the total space used in the directory tree if the **directory** option is set. In the latter case, computation of the space used is expensive, as all the files in the directory (and any sub-directories) have to be individually inspected and their sizes summed. Also, there is no interlock against two simultaneous deliveries. It is preferable to use quota mechanisms in the operating system if you can.

The value is expanded, and must then be a numerical value (decimal point allowed), optionally followed by one of the letters K or M. The expansion happens while Exim is running as root or the Exim user, before **setuid()** is called for the delivery, so files that are inaccessible to the end user can be used to hold quota values that are looked up in the expansion. When delivery fails because this quota is exceeded, the handling of the error is as for system quota failures. The value specified is not accurate to the last byte, owing to separator lines and additional headers that may get added during the delivery. See also **quota_warn_threshold**.

quota_filecount

Type: *integer*
Default: *0*

This option applies when the **directory** option is set. It limits the total number of files in the directory (compare the inode limit in system quotas). It can only be used if **quota** is also set.

quota_warn_threshold

Type: *string*
Default: *0*

This option is expanded in the same way as **quota** (see above). If the resulting value is greater than zero, and delivery of the message causes the size of the file or total space in the directory tree to cross the given threshold, then a warning message is sent. The message itself is specified by the **quota_warn_message** option, and it must start with a **To:** header line containing the recipient(s). A **Subject:** line should also normally be supplied. The **quota** option does not have to be set in order to use this option; they are independent of one another.

quota_warn_message

Type: *string*
Default: *see below*

See above for the use of this option. If it is not set when **quota_warn_threshold** is set, it defaults to

```
quota_warn_message = "\
  To: $local_part@$domain\n\
  Subject: Your mailbox\n\n\
  This message is automatically created \
  by mail delivery software.\n\n\
  The size of your mailbox has exceeded \
  a warning threshold that is\n\
  set by the system administrator.\n"
```

require_lockfile

Type: *boolean*
Default: *true*

When a lock file is being used (see **use_lockfile**) and **require_lockfile** is true, a lock file must be created before delivery can proceed. If the option is not true, failure to create a lock file is not treated as an error, though failure of the **fcntl()** locking function is. This option should always be set when delivering from more than one host over NFS. It is required to be set if the **file** option is set and **use_fcntl_lock** is not set, except when **mbx_format** is set.

retry_use_local_part

Type: *boolean*
Default: *true*

When a local delivery suffers a temporary failure, both the local part and the domain are normally used to form a key that is used to determine when next to try the address. This handles common cases such as exceeding a quota, where the failure applies to the specific local part. However, when local delivery is being used to collect messages for onward transmission by some other means, a temporary failure may not depend on the local part at all. Setting this option false causes Exim to use only the domain when handling retries for this transport.

suffix

Type: *string*
Default: *"\n"*

The string specified here is expanded and output at the end of every message. The default blank line can be suppressed by setting

```
suffix =
```

and this is usually necessary when doing batch SMTP deliveries, or delivering into individual files or MBX-format mailboxes.

use_crlf

Type: *boolean*
Default: *false*

This option causes lines to be terminated with the two-character CRLF sequence (carriage return, linefeed) instead of just a linefeed character. In the case of batched SMTP, the byte sequence written to the file is then an exact image of what would be sent down a real SMTP connection.

The contents of the **prefix** and **suffix** options are written verbatim, so must contain their own carriage return characters if these are needed. Since the default values for both **prefix** and **suffix** end with a single linefeed, their values almost always need to be changed if **use_crlf** is set.

use_fcntl_lock

Type: *boolean*
Default: *see below*

This option controls the use of the **fcntl()** function to lock a file for exclusive use when a message is being appended. It is set by default unless **use_mbx_lock** is set. Otherwise, it should be turned off only if you know that all your MUAs use lock file locking. When **use_fcntl_lock** is off, **use_lockfile** and **require_lockfile** must both be on if **mbx_format** is not set.

use_lockfile

Type: *boolean*
Default: *see below*

If this option is turned off, Exim does not attempt to create a lock file when appending to a file. Thus the only locking is by **fcntl()**. This option is set by default unless **use_mbx_lock** is set. It is not possible to turn both **use_lockfile** and **use_fcntl_lock** off, except when **mbx_format** is set. You should only turn **use_lockfile** off if you are absolutely sure that every MUA that is ever going to look at your users' mailboxes uses **fcntl()** rather than a lock file, and even then only when you are not delivering over NFS from more than one host. In order to append to an NFS file safely from more than one host, it is necessary to take out a lock *before* opening the file, and the lock file achieves this. Otherwise, even with **fcntl()** locking, there is a risk of file corruption. See also the **require_lockfile** option.

use_mbx_lock

Type: *boolean*
Default: *see below*

This option is available only if Exim has been compiled with SUPPORT_MBX set in **Local/Makefile**. Setting the option specifies that special MBX locking rules be used. It is set by default if **mbx_format** is set and none of the locking options are mentioned in the configuration. The locking rules are the same as are used by the **c-client** library that underlies Pine4 and the IMAP4 and POP daemons that come with it (see the discussion below). The rules allow for shared access to the mailbox. However, this kind of locking does not work when the mailbox is NFS mounted.

user

Type: *string*
Default: *unset*

If this option is set, it specifies the user under whose uid the delivery process is to be run, and which must be the owner of an existing file to which the message is appended. If the option is not set, a value must have been associated with the address by the director that handled it. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the transport is run, and must yield either a digit string or a name which can be looked up using **getpwnam()**. When **getpwnam()** is used, either at start-up time or later, the group id value associated with the user is taken as the value to be used if the **group** option is not set.

15.2 Operational details for appending

Before appending to a file, Exim proceeds as follows:

- If the name of the file is **/dev/null**, no action is taken, and a success return is given.
- If any directories on the file's patch are missing, Exim creates them if the **create_directory** option is set. A created directory's mode is given by the **directory_mode** option.
- If **use_lockfile** is set, a lock file is built in a way that will work reliably over NFS, as follows:
  - Create a 'hitching post' file whose name is that of the lock file with the current time, primary host name, and process id added, by opening for writing as a new file. If this fails with an access error, the message is frozen unless **require_lockfile** is false. Otherwise delivery is deferred.
  - Close the hitching post file, and hard link it to the lock file name.
  - If the call to **link()** succeeds, creation of the lock file has succeeded. Unlink the hitching post name.
  - Otherwise, use **stat()** to get information about the hitching post file, and then unlink hitching post name. If the number of links is exactly two, creation of the lock file succeeded but something (for example, an NFS server crash and restart) caused this fact not to be communicated to the **link()** call.
  - If creation of the lock file failed, wait for **lock_interval** and try again, up to **lock_retries** times. However, since any program that writes to a mailbox should complete its task very quickly, it is reasonable to time out old lock files that are normally the result of user agent and system crashes. If an existing lock file is older than **lockfile_timeout** Exim attempts to unlink it before trying again.
- A call is made to **lstat()** to discover whether the main file exists, and if so, what its characteristics are. If **lstat()** fails for any reason other than non-existence, delivery is deferred.
- If the file does exist and is a symbolic link, delivery is deferred and the message is frozen, unless the **allow_symlinks** option is set, in which case the ownership of the link is checked, and then **stat()** is called to find out about the real file, which is then subjected to the checks below. The check on the top-level link ownership prevents one user creating a link for another's mailbox in a sticky directory, though allowing symbolic links in this case is definitely not a good idea. If there is a chain of symbolic links, the intermediate ones are not checked.
- If the file already exists but is not a regular file, or if the file's owner and group (if the group is being checked – see **check_group** above) are different from the user and group under which the delivery is running, delivery is deferred, and the message is frozen.
- If the file's permissions are more generous than specified, they are reduced. If they are insufficient, delivery is deferred, and the message is frozen, unless **mode_fail_narrower** is set false.

- The file’s inode number is saved, and it is then opened for appending. If this fails because the file has vanished, **appendfile** behaves as if it hadn’t existed (see below). If the open failure is EWOULDBLOCK, just defer delivery; otherwise defer and freeze the message.
- If the file is opened successfully, check that the inode number hasn’t changed, that it is still a regular file, and that the owner and permissions have not changed. If anything is wrong, defer and freeze the message.
- If the file did not exist originally, defer delivery and freeze the message if the **file_must_exist** option is set. Otherwise, check that the file is being created in a permitted directory if the **create_file** option is set (deferring and freezing on failure), and then open for writing as a new file, with the O_EXCL and O_CREAT options, except when dealing with a symbolic link (the **allow_symlinks** option must be set). In this case, which can happen if the link points to a non-existent file, the file is opened for writing using O_CREAT but not O_EXCL, because that prevents link following.
- If opening fails because the file exists, obey the tests given above for existing files. However, to avoid looping in a situation where the file is being continuously created and destroyed, the exists/not-exists loop is broken after 10 repetitions, and the message is then frozen.
- If opening fails with any other error, defer delivery.
- Once the file is open, unless both **use_fcntl_lock** and **use_mbx_lock** are false, it is locked using **fcntl()**. In the former case, an exclusive lock is requested, while in the latter, Exim takes out a shared lock on the open file, and an exclusive lock on the file whose name is

  `/tmp/.`<*device-number*>`.`<*inode-number*>

  using the device and inode numbers of the open mailbox file, in accordance with the MBX locking rules. If locking fails, the file is closed, Exim waits for **lock_interval** and then goes back and re-opens it as above and tries to lock it again. This happens up to **lock_retries** times, after which the delivery is deferred.

At the end of delivery, Exim closes the file (which releases the **fcntl()** lock) and then deletes the lock file if one was created.

15.3 Operational details for delivery to a new file

When the **directory** option is set, each message is delivered into a newly-created file or set of files. No locking is required while writing the message, so the various locking options of the transport are ignored. The ‘From’ line that by default separates messages in a single file is not normally needed, nor is the escaping of message lines that start with ‘From’, and there is no need to ensure a newline at the end of each message. Consequently, the default settings in **appendfile** need changing as follows:

```
no_from_hack
prefix=""
suffix=""
```

There are three different ways in which delivery to individual files can be done, depending on the settings of the **maildir_format** and **mailstore_format** options. Note that code to support maildir and mailstore formats is not included in the binary unless SUPPORT_MAILDIR or SUPPORT_MAILSTORE, respectively, are set in **Local/Makefile**.

In all three cases an attempt is made to create the directory and any necessary sub-directories if they do not exist, provided that the **create_directory** option is set (the default). A created directory’s mode is given by the **directory_mode** option. If creation fails, or if the **create_directory** option is not set when creation is required, then the delivery is deferred.

- If neither **maildir_format** nor **mailstore_format** is set, a single new file is created directly in the named directory. For example, when delivering messages into files using the **bsmtp** option (see section 42.7), a setting such as

  ```
  directory = /var/bsmtp/${host}
  ```

might be used. A message is written to a file with a temporary name, which is then renamed when the delivery is complete. The final name is constructed from the time and the file's inode number, and starts with the letter 'q' for compatibility with **smail**.

- If the **maildir_format** option is true, Exim delivers each message by writing it to a file whose name is **tmp/<*time*>.<*pid*>.<*host*>** in the given directory, and then renaming it into the **new** sub-directory if all goes well.

  Before opening the temporary file, Exim calls **stat()** on its name. If any response other than ENOENT (does not exist) is given, it waits 2 seconds and tries again, up to **maildir_retries** times.

  If **maildir_tag** is set, the string is expanded for each delivery. If the expansion is forced to fail, the tag is ignored, but a non-forced failure causes delivery to be deferred. Each maildir file that is created has a colon followed by the expanded string added to its name. The tag is restricted to the alphanumeric characters plus full stop, comma, colon, hyphen, and underscore. Any other characters in the string are ignored; if the resulting string is empty, no tag is added. If the tag takes the length of the name to the point where the test **stat()** call fails with ENAMETOOLONG, then the tag is dropped and the maildir file is created with no tag.

- If the **mailstore_format** option is true, each message is written as two files in the given directory. A unique base name is constructed from the message id and the current delivery process, and the files that are written use this base name plus the suffixes **.env** and **.msg**. The **.env** file contains the message's envelope, and the **.msg** file contains the message itself.

  During delivery, the envelope is first written to a file with the suffix **.tmp**. The **.msg** file is then written, and when it is complete, the **.tmp** file is renamed as the **.env** file. Programs that access messages in mailstore format should wait for the presence of both a **.msg** and a **.env** file before accessing either of them. An alternative approach is to wait for the absence of a **.tmp** file.

  The envelope file starts with any text defined by the **mailstore_prefix** option, expanded and terminated by a newline if there isn't one. Then follows the sender address on one line, then all the recipient addresses, one per line. There can be more than one recipient only if the **batch** option is set. Finally, **mailstore_suffix** is expanded and the result appended to the file, followed by a newline if it does not end with one.

  If expansion of the prefix or suffix ends with a forced failure, it is ignored. Other expansion errors are treated as serious configuration errors, and delivery is deferred.

16. The autoreply transport

The **autoreply** transport is not a true transport in that it does not cause the message to be transmitted. Instead, it generates another mail message, usually as the result of mail filtering. A traditional 'vacation' message is the standard example.

Autoreply is implemented as a local transport so that it runs under the uid and gid of the local user and with appropriate current and home directories (see chapter 13). The parameters of the message to be sent can be specified in the configuration by the options described below, but in the common case when **autoreply** is activated as a result of filtering, none of them are normally set, because all the information is obtained from the filter file.

In an attempt to reduce the possibility of message cascades, messages created by the **autoreply** transport always take the form of delivery error messages. That is, the envelope sender field is empty.

There is a subtle difference between directing a message to a pipe transport that generates some text to be returned to the sender, and directing it to an autoreply transport. This difference is noticeable only if more than one address from the same message is so handled. In the case of a pipe, the separate outputs from the different addresses are gathered up and returned to the sender in a single message, while if **autoreply** is used, a separate message is generated for each address passed to it.

The private options of the **autoreply** transport that describe the message are used only when the address passed to it does not contain any reply information. Thus the message is specified entirely by the director or by the transport; it is never built from a mixture of options. The remaining private options (**file_optional**, **group**, **initgroups**, **mode**, **return_message**, and **user**) apply in all cases.

If any of the generic options for manipulating headers (for example, **headers_add**) are set on an **autoreply** transport, they apply to the copy of the original message that is included in the generated message when **return_message** is set. They do not apply to the generated message itself.

If the **autoreply** transport receives return code 2 from Exim when it submits the message, indicating that there were no recipients, it does not treat this as an error. This means that autoreplies sent to **$sender_address** when this is empty (because the incoming message is a delivery failure report) do not cause problems.

16.1 Private options for autor eply

bcc

Type: *string*
Default: *unset*

Specifies the addresses that are to receive 'blind carbon copies' of the message when the message is specified by the transport. The string is expanded.

cc

Type: *string*
Default: *unset*

Specifies recipients of the message and the contents of the **Cc:** header when the message is specified by the transport. The string is expanded.

file

Type: *string*
Default: *unset*

The contents of the file are sent as the body of the message when the message is specified by the transport. The string is expanded. If both **file** and **text** are set, the text string comes first.

file_expand

Type:	*boolean*
Default:	*false*

If this is set, the contents of the file named by the **file** option are subjected to string expansion as they are added to the message.

file_optional

Type:	*boolean*
Default:	*false*

If this option is true, no error is generated if the file named by the **file** option does not exist or cannot be read.

from

Type:	*string*
Default:	*unset*

The contents of the **From:** header when the message is specified by the transport. The string is expanded.

group

Type:	*string*
Default:	*unset*

If this option is set, it specifies the group under whose gid the delivery process is to be run. If it is not set, a value associated with a user may be used (see below); otherwise a value must have been associated with the address by the director which handled it. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the transport is run, and must yield either a digit string or a name which can be looked up using **getgrnam()**.

headers

Type:	*string*
Default:	*unset*

Specified additional RFC 822 headers that are to be added to the message when the message is specified by the transport. The string is expanded. Several can be given by using '\n' to separate them. There is no check on the format.

initgroups

Type:	*boolean*
Default:	*false*

If this option is true and the uid is provided by the transport, then the **initgroups()** function is called when running the transport to ensure that any additional groups associated with the uid are set up. By default no additional groups are present.

log

Type:	*string*
Default:	*unset*

This option names a file in which a record of every message sent is logged when the message is specified by the transport. The string is expanded.

mode

Type: *octal integer*
Default: *0600*

If either the log file or the ‘once’ file has to be created, this mode is used.

once

Type: *string*
Default: *unset*

This option names a DBM database in which a record of each recipient is kept when the message is specified by the transport. The string is expanded. If a potential recipient is already in the database, no message is sent by default. However, if **once_repeat** specifies a time greater than zero, the message is sent if that much time has elapsed since a message was last sent to this recipient. If **once** is unset, the message is always sent.

once_repeat

Type: *time*
Default: *0s*

See **once** above.

reply_to

Type: *string*
Default: *unset*

Specifies the contents of the **Reply-to:** header when the message is specified by the transport. The string is expanded.

return_message

Type: *boolean*
Default: *false*

If this is set, a copy of the original message is returned with the new message, subject to the maximum size set in the **return_size_limit** general configuration option.

subject

Type: *string*
Default: *unset*

The contents of the **Subject:** header when the message is specified by the transport. The string is expanded.

text

Type: *string*
Default: *unset*

This specifies a single string to be used as the body of the message when the message is specified by the transport. The string is expanded. If both **text** and **file** are set, the text comes first.

to

Type: *string*
Default: *unset*

Specifies recipients of the message and the contents of the **To:** header when the message is specified by the transport. The string is expanded.

user

Type: *string*
Default: *unset*

If this option is set, it specifies the user under whose uid the delivery process is to be run. If it is not set, a value must have been associated with the address by the director that handled it. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the transport is run, and must yield either a digit string or a name which can be looked up using **getpwnam()**. When **getpwnam()** is used, either at start-up time or later, the group id value associated with the user is taken as the value to be used if the **group** option is not set.

17. The pipe transport

The **pipe** transport is used to deliver messages via a pipe to a command running in another process. This can happen when when a director explicitly directs a message to a **pipe** transport, and also when an address is expanded via an alias, filter, or forward file that specifies a pipe command. In this case, **$local_part** contains the local part that was aliased or forwarded, while **$address_pipe** contains the text of the pipe command itself.

A **pipe** transport can also be used from a router as a pseudo-remote transport for passing messages for remote delivery by some means other than Exim.

As **pipe** is a local transport, it is always run in a separate process, normally under a non-privileged uid and gid. In the common case, these are the uid and gid belonging to the user whose **.forward** file directed the message at the pipe. In other cases the uid and gid have to be specified explicitly, either on the transport or on the director or router that handled the address. Current and 'home' directories are also controllable. See chapter 13 for details of the local delivery environment.

17.1 Returned status and data

If the command exits with a non-zero return code, the delivery is deemed to have failed, unless either the **ignore_status** option is set (in which case the return code is treated as zero), or the return code is one of those listed in the **temp_errors** option, which are interpreted as meaning 'try again later'. In this case, delivery is deferred.

If the return code is greater than 128 and the command being run is a shell script, it normally means that the script was terminated by a signal whose value is the return code minus 128.

The **return_output** option can affect the result of a pipe delivery. If it is set and the command produces any output on its standard output or standard error files, it is considered to have failed, even if it gave a zero return code or if **ignore_status** is set. The output from the command is sent as part of the delivery failure report. However, if **return_fail_output** is set, output is returned only when the command exits with a failure return code, that is, a value other than zero or a code that matches **temp_errors**.

17.2 How the command is run

By default, the command line is broken down into a command name and arguments by the **pipe** transport. The **allow_commands** and **restrict_to_path** options can be used to restrict the commands that may be run. Unquoted arguments are delimited by white space; in double-quoted arguments, backslash is interpreted as an escape character in the usual way. This does not happen for single-quoted arguments.

String expansion is applied to the command line except when it comes from a traditional **.forward** file (commands from a filter file are expanded). The expansion is applied to each argument in turn rather than to the whole line. Thus the number of arguments cannot be changed as a result of string expansion, and quotes or backslashes in inserted variables do not interact with external quoting.

Special handling takes place when an argument consists precisely of the text '`$pipe_addresses`'. This is not a general expansion variable; the only place this string is recognized is when it appears as an argument for a pipe or transport filter command. It causes each address that is being handled to be inserted in the argument list at that point *as a separate argument*. This avoids any problems with spaces or shell metacharacters, and is of use when a **pipe** transport is handling groups of addresses in a batch (see the **batch** option below).

The resulting command is then run in a subprocess directly from the transport, *not* under a shell, with the message supplied on the standard input, and the standard output and standard error both connected to a single pipe that is read by Exim. The **max_output** option controls how much output the command may produce, and the **return_output** and **return_fail_output** options control what is done with it.

Not running the command under a shell (by default) lessens the security risks in cases when a command from a user's filter file is built out of data that was taken from an incoming message. If a shell is required, it can of course be explicitly specified as the command to be run. However, there are circumstances where existing commands (for example, in **.forward** files) expect to be run under a shell and cannot easily be modified. To allow for these cases, there is an option called **use_shell**, which changes the way the **pipe** transport works. Instead of breaking up the command line as just described, it expands it as a single string and passes the result to **/bin/sh**. The **restrict_to_path** option and the **$pipe_addresses** facility cannot be used with **use_shell**, and the whole mechanism is inherently less secure.

17.3 Environment variables

The following environment variables are set up when the command is invoked:

`DOMAIN`	the local domain of the address
`HOME`	the 'home' directory – see below
`HOST`	the host name when called from a router
`LOCAL_PART`	see below
`LOGNAME`	see below
`MESSAGE_ID`	the message's id
`PATH`	as specified by the **path** option below
`QUALIFY_DOMAIN`	the configured qualification domain
`SENDER`	the sender of the message
`SHELL`	/bin/sh
`USER`	see below

The **environment** option can be used to add additional variables to this environment.

When a **pipe** transport is called directly from (for example) a **smartuser** director, then LOCAL_PART is set to the local part of the address. When it is called as a result of a forward or alias expansion, LOCAL_PART is set to the local part of the address that was expanded. LOGNAME and USER are set to the same value as LOCAL_PART for compatibility with other MTAs.

HOST is set only when a **pipe** transport is called from a router as a pseudo-remote transport (for example, for handling batched SMTP). It is set to the first host name specified by the router (if any).

If the transport's **home_directory** option is set, then its value is used for the HOME environment variable. Otherwise, certain directors may set a home directory value, as described in chapter 13.

17.4 Private options for pipe

allow_commands

Type: *string*
Default: *unset*

The string is expanded, and then is interpreted as a colon-separated list of permitted commands. If **restrict_to_path** is not set, then the only commands permitted are those in the **allow_commands** list. They need not be absolute paths; the path option is still used for relative paths. If **restrict_to_path** is set with **allow_commands**, then the command must either be in the **allow_commands** list, or a name without any slashes that is found on the path. In other words, if neither **allow_commands** nor **restrict_to_path** is set, there is no restriction on the command, but otherwise only commands that are permitted by one or the other are allowed. For example, if

```
allow_commands = /usr/ucb/vacation
```

and **restrict_to_commands** is not set, the only permitted command is **/usr/ucb/vacation**. The **allow_commands** option may not be set if **use_shell** is set.

batch

Type: *string*
Default: *"none"*

Normally, each address that is directed or routed to a **pipe** transport is handled separately. In special cases it may be desirable to handle several addresses at once, for example, when passing a message with several addresses to a different mail regime (for example, UUCP). If this option is set to the string 'domain', then all addresses with the same domain that are directed or routed to the transport are handled in a single delivery. If it is set to 'all' then multiple domains are batched. The list of addresses is included in the **Envelope-to:** header if **envelope_to_add** is set (see below). The addresses can also be set up as separate arguments to the pipe command by means of the specially-recognized argument **$pipe_addresses** (see above). Otherwise, the only difference between this option and **bsmtp** is the inclusion of SMTP command lines in the output for **bsmtp**.

batch_max

Type: *integer*
Default: *100*

This limits the number of addresses that can be handled in a batch, and applies to both the **batch** and the **bsmtp** options.

bsmtp

Type: *string*
Default: *"none"*

This option is used to set up a **pipe** transport as a pseudo-remote transport for delivering messages in batch SMTP format for onward transmission by some non-Exim means. It is usually necessary to suppress the default settings of the **prefix** and **suffix** options when using batch SMTP. The value of the option must be one of the strings 'none', 'one', 'domain', or 'all'. The first of these turns the feature off. A full description of the batch SMTP mechanism is given in section 42.7. When **bstmp** is set, the **batch** option automatically takes the same value. See also the **use_crlf** option.

bsmtp_helo

Type: *boolean*
Default: *false*

When this option is set, a HELO line is added to the output at the start of each message written in batch SMTP format. Some software that reads batch SMTP is unhappy without this.

command

Type: *string*
Default: *unset*

This option need not be set when **pipe** is being used to deliver to pipes obtained from address expansions (usually under the instance name **address_pipe**). In other cases, the option must be set, to provide a command to be run. It need not yield an absolute path (see the **path** option below). The command is split up into separate arguments by Exim, and each argument is separately expanded. Both single and double quotes are recognized. In double-quoted arguments, backslash is an escape character in the usual way. If a shell is required, it must be explicitly requested, as the command is not run under a shell by default.

current_directory

Type: *string*
Default: *unset*

If this option is set, it specifies the directory to make current when running the delivery process. The string is expanded at the time the transport is run. If this is not set, the current directory is taken from data associated with the address. See chapter 13 for full details of the local delivery environment.

environment

Type: *string*
Default: *unset*

This option is used to add additional variables to the environment in which the command runs (see section 17.3 for the default list). Its value is a string which is expanded, and then interpreted as a colon-separated list of environment settings of the form '<*name*>=<*value*>'.

freeze_exec_fail

Type: *boolean*
Default: *false*

Failure to exec the command in a pipe transport is by default treated like any other failure while running the command. However, if **freeze_exec_fail** is set, failure to exec is treated specially, and causes the message to be frozen, whatever the setting of **ignore_status**.

from_hack

Type: *boolean*
Default: *false*

If this option is true, lines in the body of the message that start with the string 'From ' are modified by adding a right angle-bracket at their start. This is necessary for traditional BSD-format mailboxes, where such lines might otherwise indicate the start of a new message.

group

Type: *string*
Default: *unset*

If this option is set, it specifies the group under whose gid the delivery process is to be run. If it is not set, a value associated with a user may be used (see below); otherwise a value must have been associated with the address by the director which handled it. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the transport is run, and must yield either a digit string or a name which can be looked up using **getgrnam()**.

home_directory

Type: *string*
Default: *unset*

If this option is set, its expanded value is used to set the HOME environment variable before running the command. This overrides any value that is set by the director. If no current directory is supplied by the director or the transport, the home directory value is used for that as well. See chapter 13 for details of the local delivery environment.

ignore_status

Type: *boolean*
Default: *false*

If this option is true, the status returned by the subprocess that is set up to run the command is ignored, and Exim behaves as if zero had been returned. Otherwise, a non-zero status causes an error return from the transport unless the value is EX_TEMPFAIL, which causes the delivery to be deferred and tried again later.

initgroups

Type: *boolean*
Default: *false*

If this option is true and the uid for the local delivery is specified by the **user** option, then the **initgroups()** function is called when running the transport to ensure that any additional groups associated with the uid are set up.

log_defer_output

Type: *boolean*
Default: *false*

If this option is set and the status returned by the command is EX_TEMPFAIL and any output was produced, the first line of it is written to the main log.

log_fail_output

Type: *boolean*
Default: *false*

If this option is set and the command returns any output and also ends with a return code that is neither zero nor EX_TEMPFAIL, the first line of output is written to the main log.

log_output

Type: *boolean*
Default: *false*

If this option is set and the command returns any output, the first line of output is written to the main log, whatever the return code.

max_output

Type: *integer*
Default: *20K*

This specifies the maximum amount of output that the command may produce on its standard output and standard error file combined. If the limit is exceeded, the process running the command is killed. This is intended as a safety measure to catch runaway processes. The limit is applied whether any **return_output** option is set or not. Because of buffering effects, the amount of output may exceed the limit by a small amount before Exim notices.

path

Type: *string-list*
Default: *"/usr/bin"*

This option specifies the string that is set up in the PATH environment variable of the subprocess. If the **command** option does not yield an absolute path name, the command is sought in the PATH directories, in the usual way.

pipe_as_creator

Type: *boolean*
Default: *false*

If **user** is not set and this option is true, then the delivery process is run under the uid that was in force when Exim was originally called to accept the message. If the group id is not otherwise set (via the **group** option above, or by the director that processed the address), then the gid that was in force when Exim was originally called to accept the message is used. Setting this option may be necessary in order to get some free-standing local delivery agents to work correctly. Note, however, that the **never_users** configuration option overrides.

prefix

Type: *string*
Default: *see below*

The string specified here is expanded and output at the start of every message. The default is the same as for the **appendfile** transport, namely

```
prefix = "From ${if def:return_path{$return_path}{MAILER-DAEMON}}\
  ${tod_bsdinbox}\n"
```

This is required by the commonly used **/usr/ucb/vacation** program, but it must *not* be present if delivery is to the Cyrus IMAP server, or to the **tmail** local delivery agent. The prefix can be suppressed by setting

```
prefix =
```

This is also usually necessary when doing batch SMTP deliveries.

restrict_to_path

Type:	*boolean*
Default:	*false*

When this option is set, any command name not listed in **allow_commands** must contain no slashes. The command is searched for only in the directories listed in the **path** option. This option is intended for use in the case when a pipe command has been generated from a user's **.forward** file. This is usually handled by a **pipe** transport called **address_pipe**.

retry_use_local_part

Type:	*boolean*
Default:	*true*

When a local delivery suffers a temporary failure, both the local part and the domain are normally used to form a key that is used to determine when next to try the address. This handles common cases such as exceeding a quota, where the failure applies to the specific local part. However, when local delivery is being used to collect messages for onward transmission by some other means, a temporary failure may not depend on the local part at all. Setting this option false causes Exim to use only the domain when handling retries for this transport.

return_fail_output

Type:	*boolean*
Default:	*false*

If this option is true, and the command produced any output and ended with a return code other than zero or EX_TEMPFAIL, the output is returned in the delivery error message. However, if the message has a null sender (that is, it is a delivery error message), output from the command is discarded.

return_output

Type:	*boolean*
Default:	*false*

If this option is true, and the command produced any output, the delivery is deemed to have failed whatever the return code from the command, and the output is returned in the delivery error message. Otherwise, the output is just discarded. However, if the message has a null sender (that is, it is a delivery error message), output from the command is always discarded, whatever the setting of this option.

suffix

Type:	*string*
Default:	*"\n"*

The string specified here is expanded and output at the end of every message. The default is the same as for the **appendfile** transport. It can be suppressed by setting

```
suffix =
```

and this is usually necessary when doing batch SMTP deliveries.

temp_errors

Type: *string*
Default: *see below*

This option contains a colon-separated list of numbers. If **ignore_status** is false and the command exits with a return code that matches one of the numbers, the failure is treated as temporary and the delivery is deferred. The default setting contains the codes defined by EX_TEMPFAIL and EX_CANTCREAT in **sysexits.h**. If Exim is compiled on a system that does not define these macros, it assumes values of 75 and 73, respectively.

timeout

Type: *time*
Default: *1h*

If the command fails to complete within this time, it is killed. This normally causes the delivery to fail. A zero time interval specifies no timeout. In order to ensure that any subprocesses created by the command are also killed, Exim makes the initial process a process group leader, and kills the whole process group on a timeout. However, this can be defeated if one of the processes starts a new process group.

umask

Type: *octal integer*
Default: *022*

This specifies the umask setting for the subprocess that runs the command.

use_crlf

Type: *boolean*
Default: *false*

This option causes lines to be terminated with the two-character CRLF sequence (carriage return, linefeed) instead of just a linefeed character. In the case of batched SMTP, the byte sequence written to the pipe is then an exact image of what would be sent down a real SMTP connection.

The contents of the **prefix** and **suffix** options are written verbatim, so must contain their own carriage return characters if these are needed. Since the default values for both **prefix** and **suffix** end with a single linefeed, their values almost always need to be changed if **use_crlf** is set.

use_shell

Type: *boolean*
Default: *false*

If this option is set, it causes the command to be passed to **/bin/sh** instead of being run directly from the transport as described in section 17.2. This is less secure, but is needed in some situations where the command is expected to be run under a shell and cannot easily be modified. The **allow_commands** and **restrict_to_path** options, and the '$pipe_addresses' facility are incompatible with **use_shell**. The command is expanded as a single string, and handed to **/bin/sh** as data for its **-c** option.

user

Type: *string*
Default: *unset*

If this option is set, it specifies the user under whose uid the delivery process is to be run. If it is not set, a value must have been associated with the address by the director that handled it. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the transport is run, and must yield either a digit string or a name which can be looked up using **getpwnam()**. When **getpwnam()** is used, either at start-up time or later, the group id value associated with the user is taken as the value to be used if the **group** option is not set.

17.5 Using an external local delivery agent

The **pipe** transport can be used to pass all messages that require local delivery to a separate local delivery agent such as **procmail**. When doing this, care must be taken to ensure that the pipe is run under an appropriate uid and gid. In some configurations one wants this to be a uid that is trusted by the delivery agent to supply the correct sender of the message. It may be necessary to recompile or reconfigure the delivery agent so that it trusts an appropriate user. The following is an example transport and director configuration for **procmail**:

```
# transport
procmail_pipe:
  driver = pipe
  command = "/opt/local/bin/procmail -d ${local_part}"
  from_hack
  user = exim

# director
procmail:
  driver = localuser
  transport = procmail_pipe
```

In this example, the pipe is run as the user **exim**, assuming that **procmail** trusts that user. Alternatively, you can run the delivery in a special group, but neither of these are mandatory. If you don't specify either a **group** or a **user** option, then the pipe command is run as the local user. The home directory is the user's home directory by default.

Note that the command that the pipe transport runs does *not* begin with

```
IFS=" "
```

as shown in the **procmail** documentation, because Exim does not by default use a shell to run pipe commands.

The next example shows a transport and a director for a system where local deliveries are handled by the Cyrus IMAP server.

```
# transport
local_delivery_cyrus:
  driver = pipe
  command = "/usr/cyrus/bin/deliver \
            -m ${substr_1:${local_part_suffix}} -- ${local_part}"
  user = cyrus
  group = mail
  return_output
  log_output
  prefix =
  suffix =

# director
local_user_cyrus:
  driver = localuser
  suffix = .*
  transport = local_delivery_cyrus
```

Note the unsetting of **prefix** and **suffix**, and the use of **return_output** to cause any text written by Cyrus to be returned to the sender.

18. The smtp transport

The **smtp** transport delivers messages over TCP/IP connections using the SMTP protocol. The list of hosts to try can either be taken from the address that is being processed, or specified explicitly for the transport. Timeout and retry processing (see chapter 31) is applied to each IP address independently. The private options are as follows:

allow_localhost

Type:	*boolean*
Default:	*false*

When a host specified in **hosts** or **fallback_hosts** (see below) turns out to be the local host, or is listed in **hosts_treat_as_local**, Exim freezes the message by default. However, if **allow_localhost** is set, it goes on to do the delivery anyway. This should be used only in special cases when the configuration ensures that no looping will result (for example, a differently configured Exim is listening on the SMTP port).

batch_max

Type:	*integer*
Default:	*0*

This controls the maximum number of separate message deliveries that can take place over a single TCP/IP connection. If the value is zero, there is no limit.

When a message has been successfully delivered over a TCP/IP connection, Exim looks in its hints database to see if there are any other messages awaiting a connection to the same host. If there are, a new delivery process is started for one of them, and the current TCP/IP connection is passed on to it. The new process may in turn create yet another process. Each time this happens, a sequence counter is incremented, and if it ever gets to the (non-zero) **batch_max** value, no further messages are sent on the same TCP/IP connection.

For testing purposes, this value can be overridden by the **-oB** command line option.

command_timeout

Type:	*time*
Default:	*5m*

This sets a timeout for receiving a response to an SMTP command that has been sent out. It is also used when waiting for the initial banner line from the remote host. Its value must not be zero.

connect_timeout

Type:	*time*
Default:	*0s*

This sets a timeout for the **connect()** function, which sets up a TCP/IP call to a remote host. A setting of zero allows the system timeout (typically several minutes) to act. To have any effect, the value of this option must be less than the system timeout.

data_timeout

Type:	*time*
Default:	*5m*

This sets a timeout for the transmission of each block in the data portion of the message. As a result, the overall timeout for a message depends on the size of the message. Its value must not be zero.

delay_after_cutoff

Type: *boolean*
Default: *true*

This option controls what happens when all remote IP addresses for a given domain have been inaccessible for so long that they have passed their retry cutoff times.

In the default state, if the next retry time has not been reached for any of them, the address is bounced without trying any deliveries. In other words, Exim delays retrying an IP address after the final cutoff time until a new retry time is reached, and can therefore bounce an address without ever trying a delivery, when machines have been down for a long time. Some people are unhappy at this prospect, so...

If **delay_after_cutoff** is set false, Exim behaves differently. If all IP addresses are past their final cutoff time, then Exim tries to deliver to those IP addresses that have not been tried since the message arrived. If there are none, of if they all fail, the address is bounced. In other words, it does not delay when a new message arrives, but immediately tries those expired IP addresses that haven't been tried since the message arrived. If there is a continuous stream of messages for the dead hosts, unsetting **delay_after_cutoff** means that there will be many more attempts to deliver to them.

dns_qualify_single

Type: *boolean*
Default: *true*

If the **hosts** or **fallback_hosts** option is being used and names are being looked up in the DNS, then the option to cause the resolver to qualify single-component names with the local domain is set.

dns_search_parents

Type: *boolean*
Default: *false*

If the **hosts** or **fallback_hosts** option is being used and names are being looked up in the DNS, then the resolver option to enable the searching of parent domains is set. Many resolvers default this option to be on, but its use in resolving mail addresses has caused problems in cases where wildcard MX records exist, so the default was changed to false in Exim version 1.80.

fallback_hosts

Type: *string-list*
Default: *unset*

String expansion is not applied to this option. The argument must be a colon-separated list of host names or IP addresses. Fallback hosts can also be specified on routers and directors which then associate them with the addresses they process; as for the **hosts** option, **fallback_hosts** specified on the transport is used only if the address does not have its own associated fallback host list.

If Exim is unable to deliver to any of the hosts for a particular address, and the errors are not permanent rejections, the address is put on a separate transport queue with its host list replaced by the fallback hosts, unless the address was routed via MX records and the current host was in the original MX list. In that situation, the fallback host list is not used.

Once normal deliveries are complete, the fallback queue is delivered by re-running the same transports with the new host lists. If several failing addresses have the same fallback hosts (and **max_rcpt** permits it), a single copy of the message is sent.

The resolution of the host names on the fallback list is controlled by the **gethostbyname()** and **mx_domains** options, as for the **hosts** option. Fallback hosts apply both to cases when the host list comes with the address and when it is taken from **hosts**. This option provides a 'use a smart host only if delivery fails' facility.

final_timeout

Type: *time*
Default: *10m*

This is the timeout that applies while waiting for the response to the final line containing just '.' that terminates a message. Its value must not be zero.

gethostbyname

Type: *boolean*
Default: *false*

If this option is true when the **hosts** and/or **fallback_hosts** options are being used, names are looked up using **gethostbyname()** instead of using the DNS with MX processing. Of course, **gethostbyname()** may in fact use the DNS to look up A (but not MX) records, but it may also consult other sources of information such as **/etc/hosts**.

hosts

Type: *string-list*
Default: *unset*

Hosts are associated with an address by a router such as **lookuphost**, which finds the hosts by looking up the address domain in the DNS. However, addresses can be passed to the **smtp** transport by any router or director, not all of which provide an associated host list. This option specifies a list of hosts which are used if the address being processed does not have any hosts associated with it, or if the **hosts_override** option is set.

The string is first expanded, before being interpreted as a colon-separated list of host names or IP addresses. Names are looked up either in the DNS (using MX processing) or using **gethostbyname()**, depending on the setting of the **gethostbyname** option. When Exim is compiled with IPv6 support, if a host that is looked up in the DNS has both A and AAAA records, all the addresses are used. See **README.IPV6** for general information about IPv6 support.

This option is typically used in association with a **smartuser** director that wants to direct messages to a particular host or hosts. The given hosts are tried in order, subject to their retry status. This option is ignored when the address has been routed by a router that supplies a host list (for example, **lookuphost**), unless **hosts_override** is set.

hosts_override

Type: *boolean*
Default: *false*

If this option is set and the **hosts** option is also set, then any hosts that are attached to the address are ignored, and instead the hosts specified by the **hosts** option are always used.

interface

Type: *string*
Default: *unset*

This option specifies which interface to bind to when making an outgoing SMTP call. The string must be an IP address, for example:

```
interface = 123.123.123.123
```

If **interface** is not set, the system's IP functions choose which interface to use if there is more than one. In an IPv6 system, the type of interface specified must be of the same kind as the address to which the call is being made. If not, it is ignored.

keepalive

Type: *boolean*
Default: *true*

This option controls the setting of SO_KEEPALIVE on outgoing socket connections. This causes the kernel periodically to send some OOB (out-of-band) data on idle connections. The option is provided for symmetry with the global **smtp_accept_keepalive** option that has the same effect on incoming SMTP connections.

max_rcpt

Type: *integer*
Default: *100*

This option limits the number of RCPT commands that are sent in a single SMTP message transaction. Each set of addresses is treated independently, and so can cause parallel connections to the same host if **remote_max_parallel** permits this.

multi_domain

Type: *boolean*
Default: *true*

When this option is set, the **smtp** transport can handle a number of addresses containing a mixture of different domains provided they all resolve to the same list of hosts. Turning the option off restricts the transport to handling only one domain at a time. This is useful if you want to use **$domain** in an expansion for the transport, because it is set only when there is a single domain involved in a remote delivery.

mx_domains

Type: *domain-list*
Default: *unset*

If the **hosts** or **fallback_hosts** options are being used and names are being looked up in the DNS, that is, the **gethostbyname** option is *not* set, then any domain name that matches this list is required to have an MX record; an A record is not sufficient.

port

Type: *string*
Default: *"smtp"*

This option specifies the TCP/IP port that is used to send the message. If it begins with a digit it is taken as a port number; otherwise it is looked up using **getservbyname()**.

retry_include_ip_address

Type: *boolean*
Default: *true*

Exim normally includes both the host name and the IP address in the key it constructs for indexing retry data after a temporary delivery failure. This means that when one of several IP addresses for a host is failing, it gets tried periodically (controlled by the retry rules), but use of the other IP addresses is not affected.

However, in some dialup environments hosts are assigned a different IP address each time they connect. In this situation the use of the IP address as part of the retry key leads to undesirable behaviour. Setting this option false causes Exim to use only the host name. This should normally be done on a separate instance of the **smtp** transport, set up specially to handle the dialup hosts.

serialize_hosts

Type:	*host-list*
Default:	*unset*

Because Exim operates in a distributed manner, if several messages for the same host arrive at around the same time, more than one simultaneous connection to the remote host can occur. This is not usually a problem except when there is a slow link between the hosts. In that situation it may be helpful to restrict Exim to one connection at a time. This can be done by setting **serialize_hosts** to match the relevant hosts.

Exim implements serialization by means of a hints database in which a record is written whenever a process connects to one of the restricted hosts, and deleted when the connection is completed. Obviously there is scope for records to get left lying around if there is a system or program crash. To guard against this, Exim ignores any records that are more than six hours old.

However, if you set up any serialization, you should also arrange to delete the hints database whenever your system reboots. The names of the files all start with **serialize-**<*transport name*> and they are kept in the **spool/db** directory. There may be one or two files per serialized transport, depending on the type of DBM in use.

service

Type:	*string*
Default:	*"smtp"*

This option is a synonym for the **port** option.

size_addition

Type:	*integer*
Default:	*1024*

If a remote SMTP server indicates that it supports the SIZE option of the MAIL command, Exim uses this to pass over the message size at the start of an SMTP transaction. It adds the value of **size_addition** to the value it sends, to allow for headers and other text that may be added during delivery by configuration options or in a transport filter. It may be necessary to increase this if a lot of text is added to messages.

Alternatively, if the value of **size_addition** is set negative, it disables the use of the SIZE option altogether.

19. Common generic options for directors and routers

Directors and routers have sufficiently many generic options in common to make it worth documenting them jointly in this chapter, to save duplication. Any of these options can be used on any director or router. Subsequent chapters describe the generic options that are specific either to directors or to routers.

condition

Type: *string*
Default: *unset*

This option specifies a test that has to succeed for the driver to be called. The string is expanded, and if the result is a forced failure or an empty string or one of the strings '0' or 'no' or 'false' (checked without regard to the case of the letters), the driver is not run. This provides a means of applying special-purpose conditions to the running of directors and routers. The **$home** variable is available in the expansion for directors that set it up. If the expansion fails, it causes Exim to panic. Some of the other options below are common special cases that could in fact be specified using **condition**.

debug_print

Type: *string*
Default: *unset*

If this option is set and debugging is enabled (see **-d**, **-v**, and **debug_level**), then the string is expanded and included in the debugging output. This is to help with checking out the values of variables and so on when debugging driver configurations. For example, if a **condition** option appears not to be working, **debug_print** could be used to output the variables it references. The output happens after checks for **domains**, **local_parts**, **suffix** and **prefix**, but before checking **require_files** and **condition**. A newline is added to the text if it does not end with one.

domains

Type: *domain-list*
Default: *unset*

If this option is set, the string is expanded, and is then interpreted as a colon-separated list. Because of the expansion, if any of the items contain backslash or dollar characters, they must be escaped with a backslash. This applies in particular to any query-style lookup that uses the **$key** variable, because otherwise it gets expanded too early. If the string is given in quotes, backslashes have to be escaped a second time.

The driver is skipped unless the current domain matches the list. If the match is achieved by means of a file lookup, then the data that the lookup returned for the domain is placed in the **$domain_data** variable for use in string expansions of the driver's private options. For directors, this option is the means by which a host can handle several independent local domains. For routers, it can be used to reduce the use of an expensive router such as **queryprogram** by doing a preliminary plausibility check on the domain. Note that the current domain may change as routing proceeds, as a router may replace the original with a different one for subsequent routers to use.

driver

Type: *string*
Default: *unset*

This option must always be set. It specifies the name of the director or router driver.

errors_to

Type:	*string*
Default:	*unset*

Delivery errors for any addresses handled or generated by the director or router are sent to the address that results from expanding this string, if it is set, and if it verifies as valid. In other words, this option sets the value of the envelope sender address to be used for deliveries associated with the driver. If it is unset, or fails to verify, the errors address associated with the incoming address (normally the sender) is used. A typical use might be

```
errors_to = "aliasmaster"
```

The **errors_to** setting associated with an address can be overridden if it subsequently passes through other directors or routers that have their own **errors_to** settings.

fail_verify

Type:	*boolean*
Default:	*false*

Setting this option has the effect of setting both **fail_verify_sender** and **fail_verify_recipient** to the same value.

fail_verify_recipient

Type:	*boolean*
Default:	*false*

If this option is true and an address is accepted by this driver when verifying a recipient, then verification fails. This option has no effect if the **verify_recipient** option is false.

fail_verify_sender

Type:	*boolean*
Default:	*false*

If this option is true and an address is accepted by this driver when verifying a sender, then verification fails. This option has no effect if the **verify_sender** option is false.

fallback_hosts

Type:	*string-list*
Default:	*unset*

String expansion is not applied to this option. The argument must be a colon-separated list of host names or IP addresses. If a driver queues an address for a remote transport, this host list is associated with the address, and used instead of the transport's fallback host list. See the **fallback_hosts** option of the **smtp** transport for further details.

group

Type:	*string*
Default:	*see below*

If a driver queues an address for a local transport, and the transport does not specify a group, then the group given here is used when running the delivery process. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the director or router is run, and must yield either a digit string or a name which can be looked up using **getgrnam()**. For most directors and routers the default is unset, but for the **forwardfile** director with **check_local_user** set, and for the **localuser** director, the default is taken from the **passwd** file. See also **initgroups** and **user** and the discussion in chapter 13.

headers_add

Type:	*string*
Default:	*unset*

This option specifies a string of text which is expanded at directing or routing time, and associated with any addresses that are processed by the driver. If the expanded string is empty, or if the expansion is forced to fail, the option has no effect. Other expansion failures are treated as configuration errors. At transport time, for each address, all original headers listed in **headers_remove** are removed, and those specified by **headers_add** are added. It is not possible to remove headers added to an address by **headers_add**.

The expanded string must be in the form of one or more RFC 822 header lines, separated by newlines (coded as '\n' inside a quoted string). For example:

```
headers_add = "X-added-header:"
```

Exim does not check the syntax of these added headers. A newline is supplied at the end if one is not present. The text is added at the end of any existing headers, but before any headers added by the transport.

If an address passes through several directors and/or routers, any **headers_add** or **headers_remove** specifications are cumulative, and any such specifications on the transport are also honoured. Addresses with different **headers_add** or **headers_remove** settings cannot be batched.

headers_remove

Type:	*string*
Default:	*unset*

The string is expanded at directing or routing time and is then associated with any addresses that are processed by the driver. If the expansion is forced to fail, the option has no effect. Other expansion failures are treated as configuration errors. After expansion, the string must consist of a colon-separated list of header names, not including the terminating colon, for example:

```
remove_headers = "return-receipt-to:acknowledge-to"
```

It is used at transport time as described under **headers_add** above.

initgroups

Type:	*boolean*
Default:	*false*

If the driver queues an address for a local transport, and this option is true, and the uid supplied by the router or director is not overridden by the transport, then the **initgroups()** function is called when running the transport to ensure that any additional groups associated with the uid are set up. See also **group** and **user** and the discussion in chapter 13.

local_parts

Type:	*string-list*
Default:	*unset*

If this option is set, the string is expanded, and is then interpreted as a colon-separated list. Because of the expansion, if any of the items contain backslash or dollar characters, they must be escaped with a backslash. This applies in particular to any query-style lookup that uses the **$key** variable, because otherwise it gets expanded too early. If the string is given in quotes, backslashes have to be escaped a second time.

The driver is run only if the local part of the address matches the list, which is tested in the same way as a domain list and which may therefore include plain file names, file lookups, and negation. Because the string is expanded, it is possible to make it depend on the domain, for example:

```
local_parts = dbm;/usr/local/specials/$domain
```

If the match is achieved by a lookup, then the data that the lookup returned for the local part is placed in the variable **$local_part_data** for use in expansions of the driver's private options. You might use this option, for example, if you have a large number of local virtual domains, and you want to send all postmaster mail to the same place without having to set up an alias in each virtual domain:

```
postmaster:
  local_parts = postmaster
  driver = smartuser
  new_address = postmaster@real.dom.ain
```

more

Type:	*boolean*
Default:	*true*

If this option is false, then if the driver fails to handle an address, no further drivers are tried, and directing or routing fails. This applies even in the case of address verification where the driver was not run because the **verify** option was off (see section 19.1). However, if a router explicitly passes an address to the following router by means of the setting

```
self = fail_soft
```

then the setting of **more** is ignored.

require_files

Type:	*string-list*
Default:	*unset*

The value of this option is first expanded and then interpreted as a colon-separated list of strings. If the option is used on a **localuser** director, or on a **forwardfile** director that has either of the **check_local_user** or **file_directory** options set, then the expansion variable **$home** may appear in the list, referring to the home directory of the user whose name is that of the local part of the address.

If any string is empty, it is ignored. Otherwise, except as described below, each string must be a fully qualified file path, optionally preceded by '!'. The paths are passed to the **stat()** function to test for the existence of the files or directories. The driver is skipped if any paths not preceded by '!' do not exist, or if any paths preceded by '!' do exist.

The **stat()** function is normally run under the exim uid (or root if such is not defined). However, it is possible to arrange for this test to be run under a specific uid and gid (which is set by means of **seteuid()** and **setegid()**). If an item in a **require_files** list does not contain any forward slash characters, it is taken to be the user (and optional group, separated by a comma) to be used for testing subsequent files in the list. If no group is specified but the user is specified symbolically, then the gid associated with the uid is used; otherwise the gid is not changed. For example:

```
require_files = mail:/some/file
require_files = ${local_part}:${home}/.procmailrc
```

The second example works because the **require_files** string is expanded before use.

If **stat()** cannot determine whether a file exists or not, delivery of the message is deferred. This can happen when NFS-mounted filesystems are unavailable.

Sometimes **stat()** yields the error EACCES ('Permission denied'). This means that the user is not permitted to read one of the directories on the file's path. The default action is to consider this a configuration error, and delivery is deferred because the existence or non-existence of the file cannot be determined. However, in some circumstances it may be desirable to treat this condition as if the file did not exist. If the file name (or the exclamation mark that precedes the file name for non-existence) is preceded by a plus sign, then the EACCES error is treated as if the file did not exist. For example:

```
  require_files = +/some/file
```

This option provides a general mechanism for predicating the running of a director or router on the existence or non-existence of certain files or directories. A failure to expand the string, or the presence of a path within it that is not fully qualified causes a panic error. This includes forced failure, because the whole string is expanded once, before being interpreted as a list. If you want a particular variant of the expansion to specify that no files are to be checked, you should cause it to yield an empty string rather than forcing failure.

senders

Type:	*address-list*
Default:	*unset*

The values of this option and **except_senders** are expanded, and the results of the expansions must be colon-separated address lists, in the same format as used for general options like **sender_reject**. The driver is run only if the sender address matches something in the **senders** list, if set, and does not match anything in **except_senders**, if set. Using this option on a director makes it possible to implement closed mailing lists (see chapter 36).

There are issues concerning verification when the running of directors or routers is dependent on the sender. When Exim is verifying an **errors_to** setting in either **forwardfile** or **aliasfile**, it sets the sender to the null string. If using the **-bt** option to check a configuration file, it is necessary also to use the **-f** option to set an appropriate sender. For incoming mail, the sender is unset when verifying the sender, but is available when verifying any recipients. If the SMTP VRFY command is enabled, it must be used after MAIL if the sender address matters.

transport

Type:	*string*
Default:	*unset*

Some directors and routers require a transport to be supplied, except when **verify_only** is set, where it is not relevant. Others require that a transport not be supplied, and for some it is optional. The string must be the name of a configured transport instance, or an expandable string, thus allowing transports to be dynamically selected. At directing or routing time, when a driver decides to accept an address, the string is expanded, and must yield the name of an available transport. If it does not, delivery is deferred. This isn't as safe as fixed transports, whose existence is checked at initialization time.

unseen

Type:	*boolean*
Default:	*false*

Setting this option has a similar effect to the **unseen** command qualifier in filter files. It causes an address to be passed on to subsequent drivers, even if the current one succeeds in handling it, and can be used to cause copies of messages to be delivered elsewhere.

user

Type:	*string*
Default:	*see below*

If the driver queues an address for a local transport, and the transport does not specify a user, then the user given here is used when running the delivery process. If the string contains no $ characters, it is resolved when Exim starts up. Otherwise, the string is expanded at the time the director or router is run, and must yield either a digit string or a name which can be looked up using **getpwnam()**. In the latter case, the group associated with the user is used as a default for the **group** option.

For most directors and routers the default for **user** is unset, but for the **forwardfile** director with **check_local_user** set, and for the **localuser** director, the default is taken from the **passwd** file. See also **initgroups** and **group** and the discussion in chapter 13.

verify

Type: *boolean*
Default: *true*

Setting this option has the effect of setting **verify_sender** and **verify_recipient** to the same value.

verify_only

Type: *boolean*
Default: *false*

If this option is set, the driver is used only when verifying an address or testing with the **-bv** option, not when actually doing a delivery, testing with the **-bt** option, or running the SMTP EXPN command (see the **expn** generic option for directors). It can be further restricted to verifying only senders or recipients by means of **verify_sender** and **verify_recipient**.

verify_recipient

Type: *boolean*
Default: *true*

If this option is false, then this driver is skipped when verifying recipient addresses. It is usual to set it false for instances of the **smartuser** director.

verify_sender

Type: *boolean*
Default: *true*

If this option is false, then this driver is skipped when verifying sender addresses. It is usual to set it false for instances of the **smartuser** director.

19.1 Skipping directors and routers

A number of the generic options that are common to directors and routers are concerned with controlling which drivers are run in particular circumstances. They interact with each other in the following way:

If the domain and local part of an address are not in agreement with **domains** and **local_parts** (when set), or if the **condition** option fails, or if **verify_only** is set and verification is not happening, then the director or router is skipped and the next one is tried. None of the other options are inspected. Otherwise, if the **more** option is false, no subsequent drivers are ever called, except when a router explicitly passes an address that routes to the local host on to the following driver, by means of the generic **self** option or the **host_find_failed** option of the **domainlist** router. The current driver is itself called unless

- Verification is happening and its **verify_sender** or **verify_recipient** option (as appropriate) is turned off, or
- The existence or non-existence of files listed in the **require_files** option is not as expected, or
- The sender of the message is not in agreement with **senders**. This test is done after checking for file existence so that sender lists can contain references to files whose existence is tested.

In the case of directors, there are some additional conditions that are tested here (see section 20.1).

The **unseen** option causes directing or routing to continue when it would otherwise cease. This is the complementary action to **no_more**, which causes it to cease when it would otherwise continue.

The **verify**, **fail_verify**, and **verify_only** options make it possible to separate those addresses which correspond to a real delivery from those which are recognized, but which do something else if actually encountered in a message.

For example, a **smartuser** director might be used to pass all unrecognized local parts to a script that tries to generate a helpful error message, or to a different machine that might be able to handle them.

This means that no local part will ever cause a delivery failure. However, if (for example) verification of senders is taking place (the **sender_verify** main configuration option), you probably don't want <*random-local-part@your.domain*> to be accepted. The solution is to set **no_verify** or **no_verify_sender** on the **smartuser** director.

On our systems in Cambridge we can identify users whose accounts have recently been cancelled, and their mail is piped to a script which sends back a more helpful message than 'user unknown'. Verification of such local parts as senders should fail, but just setting **no_verify** on the director doesn't work, because the local part is then passed to a **localuser** director that may still find it in the password file. (Initially, cancellation just resets the password.) This is the sort of case for which **fail_verify** was invented. It makes it possible to fail a set of local parts that is defined by what a specific director matches.

20. Additional generic options for directors

The following additional generic options apply to all directors, in addition to the common generic options for both directors and routers which are described in chapter 19. Directors are concerned with addresses whose domains match something in **local_domains**, or which have been explicitly determined to be local by a router.

expn

Type: *boolean*
Default: *true*

If this option is turned off, the director is skipped when testing an address as a result of processing an SMTP EXPN command. You might, for example, want to turn it off on a director for users' **.forward** files, while leaving it on for the system alias file. The use of the SMTP EXPN command is permitted only from hosts that match the **smtp_expn_hosts** main configuration option.

This option is specific to directors because EXPN applies only to local addresses, so no address that is an argument to EXPN is ever passed to any router. When Exim is running an EXPN command, it is similar to testing an address with **-bt**. Compare VRFY, whose counterpart is **-bv**.

new_director

Type: *string*
Default: *unset*

Sometimes an administrator knows that it is pointless to reprocess addresses generated from alias or forward files with the same director again. For example, if an alias file translates real names into login ids there is no point searching the alias file again, especially if it is a large file.

The **new_director** option can be set to the name of any director instance. It causes the directing of any generated local addresses to start at the named director instead of the first director. The named director can be any configured director. This option has no effect if the director in which it is set does not generate new addresses, or if such addresses are not in local domains.

prefix

Type: *string-list*
Default: *unset*

If this option is set, the director is skipped unless the local part starts with one of the given strings, or the **prefix_optional** option is true. A limited form of wildcard is available; if the prefix begins with an asterisk, it matches the longest possible sequence of arbitrary characters at the start of the local part. An asterisk should therefore always be followed by some character that does not occur in normal local parts. Wildcarding can be used to set up multiple user mailboxes, as described in chapter 35.

While the director is running, the prefix is removed from the local part, and is available in the expansion variable **local_part_prefix**. If the director succeeds, this remains true during subsequent delivery.

The prefix facility is commonly used to handle local parts of the form **owner-something**. Another common use is to support local parts of the form **real-username** to bypass a user's **.forward** file – helpful when trying to tell a user their forwarding is broken – by placing a director like this one immediately before the director that handles **.forward** files:

```
real_localuser:
  driver = localuser
  transport = local_delivery
  prefix = real-
```

If both **prefix** and **suffix** are set for a director, both conditions must be met if not optional. Care must be taken if wildcards are used in both a prefix and a suffix on the same director. Different separator characters must be used to avoid ambiguity.

prefix_optional

Type: *boolean*
Default: *false*

See **prefix** above.

suffix

Type: *string-list*
Default: *unset*

This option operates in the same way as **prefix**, except that the local part must end (rather than start) with the given string, the **suffix_optional** option determines whether the suffix is mandatory, and the wildcard $*$ character, if present, must be the last character of the suffix. This option facility is commonly used to handle local parts of the form **something-request** and multiple user mailboxes of the form **username-foo**.

suffix_optional

Type: *boolean*
Default: *false*

See **suffix** above.

20.1 Skipping directors

Section 19.1 above describes the circumstances in which the generic options that are common to both directors and routers cause a driver to be skipped. Directors have additional generic options which impose some further condition.

The **new_director** generic option causes the directing of a generated local address to start at a particular director, thus skipping those above it for that address.

Processing of the **prefix** and **suffix** options does not happen until after the check of **local_parts** is done, so the local part that is checked at that stage is the full local part. If you want to select a director based on a partial local part, you can use a regular expression, or make use of the **condition** option to do more complicated processing (such as looking up a prefix-stripped local part in a file).

The following additional conditions, which are applied after the initial checks on the domain etc., prevent the current director from being run:

- An SMTP EXPN command is being processed and the director's **expn** option is turned off, or
- There is a prefix or suffix mismatch, or
- The address was generated by aliasing or forwarding and is identical to an ancestor address that was processed by this director. This restriction breaks addressing loops.

21. The aliasfile director

The **aliasfile** director expands local parts by consulting a file of aliases. The expansion may safely contain the same local part as the input, because a director is automatically skipped if any ancestor of a local part has the same name and was processed by that director.

The alias file can be a text file that is searched linearly, a DBM direct-access database, a NIS or NIS+ map, or any other kind of lookup supported by Exim (see chapter 6).

Unless the **locally_caseless** option has been set false, local parts are forced to lower case, and so the keys in alias files should normally be in lower case. For linearly searched files this isn't in fact necessary, because the searching is done in a case-independent manner, but it is relevant for other forms of alias lookup. The **exim_dbmbuild** utility can be used to convert a text file into a DBM database; the keys are lower-cased by default.

21.1 Alias file format

A textual alias file to be searched linearly consists of entries that start with the alias name, terminated by a colon or white space. However, a colon must be used if data for the alias starts with a colon, because white space is permitted between the alias name and its terminating colon. This is Exim's standard **lsearch** format (see chapter 6).

The remainder of the entry, up to the end of the line, consists of a list of addresses, file names, pipe commands, or certain special items (see below). The items in the list are separated by commas. The list can be continued over several lines by starting each of the continuation lines with white space. A comma is still required following an item that ends at the end of a line, because the **lsearch** lookup code removes newlines from the string it returns.

Lines in textual alias files that start with a # character are comments, and are ignored, and a # may also appear following a comma in an item list, in which case everything after the # is ignored.

Other forms of alias file (DBM, NIS, etc.) involve lookups using the local part as a key on files and databases. The value returned is a list of address, file, or pipe items separated by commas or newlines.

In all cases, the returned list is normally used exactly as it stands, but if the **expand** option is set, it is first passed through the string expansion mechanism.

By default, alias names are simple local parts such as 'postmaster', but if the **include_domain** option is set, they must contain both a local part and a domain, thus allowing aliases for more than one domain to be held in a single file.

It is possible to set up a default in an alias file, to match any incoming local part that doesn't match any other entry. This is done by making use of a lookup type name followed by an asterisk, for example **dbm*** (see chapter 6).

21.2 Types of alias item

If an item is entirely enclosed in double quotes, these are removed. Otherwise double quotes are retained because some forms of mail address require their use (but never to enclose the entire address). In the following description, 'item' refers to what remains after any surrounding double quotes have been removed. An item may safely be the same as the one currently under consideration, because any director is automatically skipped if any ancestor has the same local part and was processed by that director.

- If an item begins with '\' and the rest of the item parses as a valid RFC 822 address that does not include a domain, the item is qualified using the domain of the incoming address. The use of '\' makes a difference only if there is more than one local domain. In the absence of a leading '\', unqualified addresses are qualified using the value in **qualify_recipient**, unless **qualify_**

preserve_domain is set. It is not necessary to include '\' in aliases to prevent directing loops, because Exim has its own method of loop detection.

- An item is treated as a pipe command if it begins with '|' and does not parse as a valid RFC 822 address that includes a domain. A transport for running the command must be specified by the **pipe_transport** option. Either the director or the transport must specify a user and group under which to run the delivery.

  Either single or double quotes can be used for enclosing the individual arguments of the pipe command; no interpretation of escapes is done for single quotes. If the command contains a comma character, it is necessary to put the whole item in double quotes, for example:

  ```
  "|/some/command ready,steady,go"
  ```

  since items are terminated by commas. Do not, however, quote just the command. An item such as

  ```
  |"/some/command ready,steady,go"
  ```

  is interpreted as a pipe with a rather strange command name, and no arguments.

- An item is interpreted as a path name if it begins with '/' and does not parse as a valid RFC 822 address that includes a domain. For example,

  ```
  /home/world/minbari
  ```

  is treated as a file name, but

  ```
  /s=molari/o=babylon/@x400gate.way
  ```

  is treated as an address. For a file name, a transport must be specified using the **file_transport** option. However, if the generated path name ends with a forward slash character, it is interpreted as a directory name rather than a file name, and **directory_transport** is used instead. If it ends with two slashes, **directory2_transport** is required. This makes it possible to support two different kinds of directory delivery simultaneously.

  If a generated path is **/dev/null**, delivery to it is bypassed at a high level, and the log entry shows '**bypassed**' instead of a transport name. This avoids the need to specify a user and group, which are necessary for a genuine delivery to a file. When the file name is not **/dev/null**, either the director or the transport must specify a user and group under which to run the delivery.

- An item of the form

  `:include:`<*path name*>

  may appear in an alias file, in which case a list of further items is taken from the given file and included at that point. The items in the list are separated by commas or newlines and are not subject to expansion, even when the **expand** option is set. If this is the first item in an alias list, a colon must be used to terminate the alias name.

- Sometimes you want to throw away mail to a particular local part. An alias entry with no addresses causes Exim to generate an error, so that cannot be used. However, another special item that may appear in an alias file is

  ```
  :blackhole:
  ```

  which does what its name implies. No delivery is done for it, and no error message is generated. If this is the first item in an alias list, a colon must be used to terminate the alias name.

  This used to be more efficient than directing a message to **/dev/null** because it happens at directing time, and also there was no need to specify a user and group to run the transport process for delivery to a file. However, from Exim version 1.90 onwards **/dev/null** is recognized specially, and handled in essentially the same way as **:blackhole:**.

- An attempt to deliver to a particular local part can be deferred or forced to fail by aliasing the local part to

```
:defer:
```
or
```
:fail:
```

respectively. As this is normally the only item in an alias list, a colon must be used to terminate the alias name.

When an alias list contains such an item, it applies to the entire alias; any other items in the list are ignored (**:blackhole:** is different). Any text following **:fail:** or **:defer:** is placed in the error message. For example:

```
X.Employee:  :fail: Gone away, no forwarding address
```

In the case of an address that is being verified by the SMTP RCPT or VRFY commands, the text is included in the SMTP error response. In other cases it is included in the error message that Exim generates.

Normally the text is the rest of the alias entry – a comma does not terminate it – but a newline does act as a terminator. Newlines are not normally present in alias expansions. In **lsearch** lookups they are removed as part of the continuation process, but they may exist in other kinds of lookup and in **:include:**d files.

An alias containing **:fail:** causes an immediate failure of the incoming address, whereas **:defer:** causes the message to remain on the queue so that a subsequent delivery attempt can happen at a later time. If an address is **:defer:**red for too long, it will ultimately fail, since normal retry rules apply.

- Sometimes it is useful to use a search type with a default (see chapter 6) for aliases. However, there may be a need for exceptions to the default. These can be handled by aliasing them to

  ```
  :unknown:
  ```

  This differs from **:fail:** in that it causes **aliasfile** to pass the address on to the next director, whereas **:fail:** forces directing to fail immediately. If **:unknown:** is the first item in an alias list, a colon must be used to terminate the alias name.

21.3 Duplicate addresses

Exim removes duplicate addresses from the list to which it is delivering, so as to deliver just one copy to each address. This does not apply to deliveries directed at pipes by different immediate parent addresses, but an indirect aliasing scheme of the type

```
pipe:        |/some/command ${local_part}
localpart1: pipe
localpart2: pipe
```

does not work with a message that is addressed to both local parts, because when the second is aliased to the intermediate local part 'pipe' it gets discarded as being the same as a previously handled address. However, a scheme such as

```
localpart1: |/some/command ${local_part}
localpart2: |/some/command ${local_part}
```

does result in two different pipe deliveries, because the immediate parents of the pipes are distinct.

21.4 Repeated alias expansion

When a message cannot be delivered to all of its recipients immediately, leading to two or more delivery attempts, alias expansion is carried out afresh each time for those addresses whose children were not all previously delivered. If an alias is being used as a mailing list, this can lead to new members of the list receiving copies of old messages. The **one_time** option can be used to avoid this.

21.5 Errors in alias files

If **skip_syntax_errors** is set, a malformed address that causes a parsing error is skipped, and an entry is written to the main log. This may be useful for mailing lists that are automatically managed, but note the inherent danger. Otherwise, if an error is detected while generating the list of new addresses, the message is frozen, except for the special case of inability to open an included file, when **no_freeze_missing_include** is set. In this case, delivery is simply deferred.

21.6 Specifying transports for aliasfile

If the generic **transport** option is specified for this director, then the message is directed to that transport for any local part which is found in the file, any data in the file that is associated with the local part being ignored (so it isn't really 'aliasing' in this case). Thus the same processing can be done for any local part that is listed in the file. For example, a file containing a list of cancelled users can be used to direct messages addressed to them to a particular script.

A common use of **aliasfile** with a transport is for handling local deliveries without reference to **/etc/passwd**. Local parts are validated by using **aliasfile** to look them up in a file or database, which can also be used to hold information for use during delivery (for example, the uid to use, or the location of the mailbox). There is a sample configuration that gives more detail.

There are also a number of private options that specify transports for use in certain cases, for example, when a local part is aliased directly to a file name, or to a pipe command.

21.7 Aliasfile private options

check_ancestor

Type:	*boolean*
Default:	*false*

When this option is set, if a generated address is the same as *any* ancestor of the current address, then it is not used, but instead the current address gets passed on to subsequent directors. In the default case, this happens only if the ancestor was processed by the current director. See the **check_ancestor** option on the **forwardfile** director for more details.

current_directory

Type:	*string*
Default:	*unset*

This option associates a current directory with any address that **aliasfile** directs to a local transport. This can happen either because a transport is explicitly configured for the director, or because it generates a delivery to a file or a pipe. The option string is expanded and is set as the current directory during the delivery process, unless overridden by a setting on the transport. See chapter 13 for details of the local delivery environment.

directory_transport

Type:	*string*
Default:	*unset*

An **aliasfile** director sets up a direct delivery to a directory when a path name ending with a slash is specified as a new 'address'. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

directory2_transport

Type:	*string*
Default:	*unset*

An **aliasfile** director sets up an alternative direct delivery to a directory when a path name ending with two slashes is specified as a new ‘address’. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

errors_to

Type: *string*
Default: *unset*

This used to exist as an option specific to this director, but it is now a generic option that can be used on any director or router (see chapter 19).

expand

Type: *boolean*
Default: *false*

If this option is set true, then the text obtained by looking up the local part is passed through the string expansion mechanism before being interpreted as a list of alias items. Addresses that are subsequently added by means of the ‘include’ mechanism are *not* expanded.

file

Type: *string*
Default: *unset*

This option specifies the name of the alias file, and it must be set if **search_type** specifies a single-key lookup; if it does not, an error occurs. (For query-style lookups, **query** must be set instead.) See chapter 6 for details of different lookup styles. The string is expanded before use; if expansion fails, Exim panics. The resulting string must be an absolute path for linear search and DBM lookups. If the original string does not start with ‘/’ or ‘$’ in these cases, Exim gives a configuration error when it starts up; otherwise, if an expanded string does not begin with ‘/’ delivery is frozen.

file_transport

Type: *string*
Default: *unset*

An **aliasfile** director sets up a direct delivery to a file when a path name not ending in a slash is specified as a new ‘address’. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

forbid_file

Type: *boolean*
Default: *false*

If this option is true, then this director may not generate a new address which specifies delivery to a local file or directory. If it attempts to do so, a delivery failure occurs.

forbid_pipe

Type: *boolean*
Default: *false*

If this option is true, then this director may not generate a new address which specifies delivery to a pipe. If it attempts to do so, a delivery failure occurs.

freeze_missing_include

Type: *boolean*
Default: *true*

If a file named by the ‘include’ mechanism fails to open, delivery is frozen if this option is true. Otherwise, delivery is just deferred. Unsetting this option can be useful if included files are NFS mounted and may not always be available.

home_directory

Type: *string*
Default: *unset*

This option associates a home directory with any address that **aliasfile** directs to a local transport. This can happen either because a transport is explicitly configured for the director, or because it generates a delivery to a file or a pipe. The option string is expanded and is set as the home directory during the delivery process, unless overridden by a setting on the transport. See chapter 13 for details of the local delivery environment. This means that the expansion variable **$home** does not take on this value at directing time. In particular, it cannot be used in the **require_files** option.

include_domain

Type: *boolean*
Default: *false*

Setting this option true causes the key that is looked up to be 'local-part@domain' instead of just 'local-part'. Thus a single file can be used to hold aliases for many local domains. This option has no effect if the search type specifies a query-style lookup.

If you want include defaults for each domain in an alias file in the form

```
*@domain1:  default@domain1
*@domain2:  default@domain2
```

then you need to include '*@' in the search type (for example, **dbm*@**). See section 6.1 for details of this kind of search.

modemask

Type: *octal integer*
Default: *022*

This specifies mode bits which must not be set for the alias file. If they are set, the director fails and the message is frozen.

one_time

Type: *boolean*
Default: *false*

Sometimes the fact that Exim re-evaluates aliases each time it tries to deliver a message causes problems. This is particularly true in the case of mailing lists and so is more likely to be a problem with forward files than with alias files.

If **one_time** is set and any addresses generated by the director fail to deliver at the first attempt, the failing addresses are added to the message as 'top level' addresses, and the parent address that generated them is marked 'delivered'. Thus aliasing does not happen again at the next delivery attempt. To ensure that **aliasfile** generates only addresses (as opposed to pipe or file deliveries) **forbid_file** and **forbid_pipe** must also be set.

The original top-level address is remembered with each of the generated addresses, and is output in any log messages. However, any intermediate parent addresses are not recorded. This makes a difference to the log only if **log_all_parents** is set. It is expected that **one_time** will typically be used for mailing lists, where there is normally just one level of expansion.

optional

Type: *boolean*
Default: *false*

If the file cannot be opened because it does not exist (the ENOENT error) and this option is set, the director simply fails to match the address. Otherwise any failure to open the file causes an entry to be written to the log and delivery to be deferred.

owners

Type: *string-list*
Default: *unset*

This specifies a list of permitted owners for the alias file. If it is unset, no check on the ownership is done. If the file is not owned by a user in the list, the director fails and the message is frozen.

owngroups

Type: *string-list*
Default: *unset*

This specifies a list of permitted groups for the alias file. If it is unset, no check on the file's group is done. If the file's group is not in the list, the director fails and the message is frozen.

pipe_transport

Type: *string*
Default: *unset*

An **aliasfile** director sets up a direct delivery to a pipe when a string starting with a vertical bar character is specified as a new 'address'. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

qualify_preserve_domain

Type: *boolean*
Default: *false*

If this is set and an unqualified address (one without a domain) is generated, it is qualified with the domain of the incoming address instead of the global setting in **qualify_recipient**.

queries

Type: *string*
Default: *unset*

This option is an alternative to **query**; the two options are mutually exclusive. The difference is that **queries** contains a colon-separated list of queries, which are tried in order until one succeeds or defers, or all fail. Any colon characters actually required in an individual query must be doubled, in order that they not be treated as query separators.

query

Type: *string*
Default: *unset*

This option specifies a database query, and either it or **queries** must be set if **search_type** specifies a query-style lookup; if neither is set, an error occurs. (For single-key lookups, **file** must be set instead.) See chapter 6 for details of different lookup styles. The query is expanded before use, and would normally contain a reference to the local part. For example,

```
search_type = nisplus
query = [alias=${local_part}],mail_aliases.org_dir:expansion
```

could be used for a NIS+ lookup. Sometimes a lookup cannot be completed (for example, a NIS+ database might be inaccessible) and in this case the director causes delivery to be deferred.

rewrite

Type: *boolean*
Default: *true*

If this option is set false, addresses generated by the director are not subject to address rewriting. Otherwise, they are treated like new addresses.

search_type

Type: *string*
Default: *unset*

This option must be set to the name of a supported search type ('lsearch', 'dbm', etc.), specifying the type of data lookup. For query-style lookups, the **query** option specifies the search query, and **file** must not be set. For the other search types, **file** is required and **query** must not be set. See chapter 6 for details of the different lookup styles.

Single-key search types for **aliasfile** can be preceded by **partial-** and/or followed by ∗. The former isn't likely to be useful very often, but the latter provides a default facility. Note, however, that if two addresses in the same message provoke the use of the default, only one copy gets delivered, but any added **Envelope-to:** header contains all the original addresses. Exceptions to the default can be set up by aliasing them to **:unknown:**.

skip_syntax_errors

Type: *boolean*
Default: *false*

If **skip_syntax_errors** is set, a malformed address that causes a parsing error is skipped, and an entry is written to the main log. This may be useful for mailing lists that are automatically managed, but note the inherent danger. Exim always considers it to be an error if no addresses at all are generated, even if this option is set.

syntax_errors_text

Type: *string*
Default: *unset*

See **syntax_errors_to**.

syntax_errors_to

Type: *string*
Default: *unset*

This option applies only when **skip_syntax_errors** is set. If any addresses are skipped because of syntax errors, a mail message is sent to the address specified by **syntax_errors_to**, giving details of the failing address(es). If **syntax_errors_text** is set, its contents are expanded and placed at the head of the error message. Often it will be appropriate to set **syntax_errors_to** to be the same address as the generic **errors_to** option.

22. The forwardfile director

The **forwardfile** director can be used for two different but related operations. Its effect is to replace a local part with a list of addresses, file names, or pipe commands, taken from a single file. It gets its name from the common case where the file is in a user's home directory and is called **.forward**, but another common use is for expanding mailing lists, which are discussed in more detail in chapter 36.

A standard transport must *not* be specified for this director. That is, the generic **transport** option must not be set. A configuration error occurs if one is given. However, the special transports for handling files, pipes, and autoreplies must be set if needed.

When handling a user's **.forward** file, a uid, gid, and home directory are commonly obtained from the password file by calling **getpwnam()**. However, these may alternatively be specified by options to the director, in which case **getpwnam()** is not called.

22.1 Forward file items

The contents of the file are a list of addresses, file names, or pipe commands, separated by commas or newlines. Items that are empty are ignored. This includes items consisting solely of RFC 822 address comments. If an item is entirely enclosed in double quotes, these are removed, but otherwise double quotes are retained, because some forms of mail address require the use of double quotes, though never enclosing the whole address.

Lines starting with a # character are comments, and are ignored, and # may also appear following a comma, in which case everything between the # and the end of the line is ignored. If the file is empty, or contains only blank lines and comments, the director behaves as if it did not exist.

If a message is addressed to two or more different local parts, each of which results in an expansion that generates an identical file name or pipe command, different deliveries occur, though of course each delivery process runs with different values in the LOCAL_PART environment variable, and with different uids (in the common case). This happens only if the immediate ancestors of the pipes or files are different local parts. If several different local parts generate an intermediate alias which in turn generates a pipe or file delivery, only a single delivery is done, because the duplicate intermediate addresses are discarded.

- An address item may safely be the same local part as the one currently under consideration, because a director is automatically skipped if any ancestor has the same local part and was processed by that director. Thus a user with login name **spqr** who wants to preserve a copy of mail and also forward it somewhere else can set up a file such as

  ```
  spqr, spqr@st.else.where
  ```

  without provoking a loop. A backslash before an unqualified local part is permitted for compatibility with other mailers, but is not necessary for loop prevention. The presence or absence of a backslash does, however, make a difference when there is more than one local domain. Without a backslash, an unqualified local part is qualified with the contents of **qualify_recipient** unless **qualify_preserve_domain** is set, but if a backslash is present, the local part is always qualified with the domain of the incoming address.

  Care must be taken if there are alias names for local users. For example if the system alias file contains

  ```
  Sam.Reman: spqr
  ```

  then

  ```
  Sam.Reman, spqr@reme.else.where
  ```

  in **spqr**'s forward file fails on an incoming message addressed to **Sam.Reman**, because the **aliasfile** director does not process **Sam.Reman** the second time round, having previously done so. The forward file should really contain

```
spqr, spqr@reme.else.where
```

but because this is such a common error, the **check_ancestor** option (see below) exists to provide a way to get round it.

- An item is interpreted as a file name if it begins with '/' and does not parse as a valid RFC 822 address that includes a domain. For example,

  ```
  /home/world/shadow
  ```

  is treated as a file name, but

  ```
  /s=molari/o=babylon/@x400gate.way
  ```

  is treated as an address. For a file name, a transport must be specified using the **file_transport** option. However, if the generated path name ends with a forward slash character, it is interpreted as a directory name rather than a file name, and **directory_transport** is used instead. If it ends with two slashes, **directory2_transport** is required. This makes it possible to support two different kinds of directory delivery simultaneously.

  If an item is **/dev/null**, delivery to it is bypassed at a high level, and the log entry shows '**bypassed**' instead of a transport name. This avoids the need for a user and group, which are necessary for a genuine delivery to a file. When the file name is not **/dev/null**, either the director or the transport must specify a user and group under which to run the delivery. If **check_local_user** is set, the uid and gid from the **passwd** file are used as defaults for the generic **user** and **group** options.

- An item is treated as a pipe command if it begins with '|' and does not parse as a valid RFC 822 address that includes a domain. A transport for running the command must be specified by the **pipe_transport** option. Either the director or the transport must specify a user and group under which to run the delivery. If **check_local_user** is set, the uid and gid from the **passwd** file are used as defaults for the generic **user** and **group** options.

  Both single and double quotes can be used for enclosing individual arguments to the pipe command; no interpretation of escapes is done for single quotes. If the command contains a comma character, it is necessary to put the whole item in double quotes, for example:

  ```
  "|/some/command ready,steady,go"
  ```

  since items are terminated by commas. Do not, however, quote just the command. An item such as

  ```
  |"/some/command ready,steady,go"
  ```

  is interpreted as a pipe with a rather strange command name, and no arguments.

- Instead of an address, file name, or pipe command, an item of the form

  `:include:`<*path name*>

  may appear, in which case a list of addresses is taken from the given file and included at that point, unless the **forbid_include** option is set. There are some security considerations when such an item is included in a user's **.forward** file:

  (i) If the **seteuid()** function is being used to read the main file as a specific user (see **seteuid** below) then the included file is read as the same user.

  (ii) Otherwise Exim is running as root at this point. If **check_local_user** is set, or if an explicit directory is specified, then any included files must be within the home or given directory, and no symbolic links are permitted below the directory name.

  (iii) If neither **check_local_user** nor **directory** is set when **seteuid()** is not in use, then included files are not permitted.

22.2 Repeated forwarding expansion

When a message cannot be delivered to all of its recipients immediately, leading to two or more delivery attempts, forwarding expansion is carried out afresh each time for those addresses whose children were not all previously delivered. If a forward file is being used as a mailing list, this can lead to new members of the list receiving copies of old messages. The **one_time** option can be used to avoid this.

22.3 Errors in forward files

If **skip_syntax_errors** is set, a malformed address that causes a parsing error is skipped, and an entry is written to the main log. This may be useful for mailing lists that are automatically managed, but note the inherent danger. The option should never be set for users' **.forward** files. Otherwise, if any error is detected while generating the list of new addresses, the message is frozen, except for the special case of inability to open an included file when **no_freeze_missing_include** is set. In this case, delivery is simply deferred.

22.4 Filter files

As an alternative to treating the file as a simple list of addresses, the **forwardfile** director can be configured, by means of the **filter** option, to read a file and interpret it as a list of *filtering* instructions if it conforms to a specific format. The instructions can specify various actions such as appending the message to certain mail folders, or forwarding it to other users, predicated on the content of the message. Details of the syntax and semantics of filter files are described in a separate document entitled *Exim's User interface to mail filtering*; this is intended for use by end users. If filters are permitted to generate mail messages (see **forbid_reply**) then the **reply_transport** option must be set.

22.5 The home dir ectory

The **home** expansion variable can be used in a number of local options for **forwardfile**. Its value depends on the way the options are set up, as follows:

- If **check_local_user** is set without **file_directory**, then the user's home directory is set in the **home** expansion variable when expanding the **file** option that specifies a forward or filter file.
- If **file_directory** is set without **check_local_user**, then the expanded value of **file_directory** is set in the **home** expansion variable when expanding the **file** option. If **home** appears in the string for **file_directory**, its substitution value is the empty string.
- If both **check_local_user** and **file_directory** are set, **home** in the string for **file_directory** is the user's home directory, but **home** in the **file** option is the expanded value of **file_directory**.

If the generic **require_files** option, or any other expanded option, contains **$home**, it takes the same value as it does when expanding the **file** option, and this value is also used for **$home** if encountered in a filter file, and as the default value to pass with the address when a pipe or file delivery is generated.

22.6 Forwardfile private options

allow_system_actions

Type:	*boolean*
Default:	*false*

Setting this option permits the use of **freeze** and **fail** in filter files. This should *not* be set on the director for users' **.forward** files, but can be useful if you want to run a system-wide filter for each address, as opposed to the system filter, which runs just once per message. See chapter 41.

check_ancestor

Type: *boolean*
Default: *false*

Although this option is off by default in the code, it is set in the default configuration file for handling users' **.forward** files. It is recommended for this use of the **forwardfile** director. When set, if a generated address is the same as *any* ancestor of the current address, then it is not used, but instead the current address gets passed on to subsequent directors. This helps in the case where local part A is aliased to B, and B has a **.forward** file pointing back to A, for example: 'Joe.Bloggs' is aliased to 'jb' and **~jb/.forward** contains:

```
\Joe.Bloggs, some.other.address
```

Without the **check_ancestor** setting, either local part ('jb' or 'joe.bloggs') gets processed once by each director and so ends up as it was originally. If 'jb' is the real mailbox name, then mail to 'jb' gets delivered (having been turned into 'joe.bloggs' by the **.forward** file and back to 'jb' by the alias), while mail to 'joe.bloggs' fails. Setting **check_ancestor** on the **forwardfile** director prevents it from turning 'jb' back into 'joe.bloggs' when that was the original address.

The **aliasfile** director also has a **check_ancestor** option for use in special cases. Setting it does not have the desired effect in a conventional configuration.

check_group

Type: *boolean*
Default: *false*

The group of the file is checked only when this option is set. If **check_local_user** is set, then the user's default group is permitted; otherwise the group must be one of those listed in the **owngroups** option.

check_local_user

Type: *boolean*
Default: *true*

If this option is true, then the local part that is passed to this director is checked to ensure that it is the login of a local user by calling the **getpwnam()** function. The director fails to handle the address if it is not. In addition, when this option is true, the string specified for the **file** option is taken as relative to the user's home directory if it is not an absolute path, unless the **file_directory** option is set.

When this option is set, the local user is always one of the permitted owners of the file, and the local user's uid is used when reading the forward file if the **seteuid** option is set or if the global security setting is not 'setuid'. In addition the uid and gid read from the **passwd** file are used as defaults for the generic **user** and **group** options.

current_directory

Type: *string*
Default: *unset*

This option associates a current directory with any address that **forwardfile** directs to a local transport because it specifies a file name, pipe command, or autoreply. The option string is expanded and is set as the current directory during the delivery process, that is, at transport time, unless overridden by a setting on the transport. See chapter 13 for details of the local delivery environment.

A special string value is provided for the case when both **check_local_user** and **file_directory** are set, when **$home** contains the **file_directory** value. If **current_directory** contains the string 'check_local_user' then it is replaced by the user's home directory path.

directory_transport

Type: *string*
Default: *unset*

A **forwardfile** director sets up a direct delivery to a directory when a path name ending with a slash is specified as a new ‘address’. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

directory2_transport

Type: *string*
Default: *unset*

A **forwardfile** director sets up a direct delivery to a directory when a path name ending with two slashes is specified as a new ‘address’. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

errors_to

Type: *string*
Default: *unset*

This used to exist as an option specific to this director, but it is now a generic option that can be used on any director or router (see chapter 19).

file

Type: *string*
Default: *unset*

This option must be set. The string is expanded before use – see above for a discussion of the **home** expansion variable. If expansion fails, Exim panics. The expanded string is interpreted as a single file name, and must start with a slash character unless **check_local_user** is true, or a **file_directory** option is set. A non-absolute path is interpreted relative to the **file_directory** setting if it exists; otherwise it is interpreted relative to the user’s home directory.

If a non-absolute path is used, Exim uses the **stat()** function to check the directory before attempting to open the file therein. If the directory is inaccessible, the delivery to the current address is deferred. This distinguishes between the cases of a non-existent file (where the director cannot handle the address) and an unmounted NFS directory (where delivery should be deferred). Thus the difference between the two settings

```
file = .forward
file = $home/.forward
```

is that in the second case the directory is not checked with **stat()**.

If the file exists but is empty or contains only blank and comment lines starting with #, Exim behaves as if it did not exist, and the director fails to handle the address. Note that this is not the case when the file contains syntactically valid items that happen to yield empty addresses, for example, items containing only RFC 822 address comments.

file_directory

Type: *string*
Default: *unset*

The string is expanded before use – see above for a discussion of the **home** expansion variable. The option sets a directory path which is used if the **file** option does not specify an absolute path. This on its own is not very useful, since the directory string could just as well be prepended to the file string. However, if a separate directory is given, it is treated like a directory obtained from **check_local_user**, and its existence is tested before trying to open the file. If the directory appears not to exist, delivery is deferred. Thus, a setting such as

```
  file_directory = /usr/forwards
  file = ${local_part}.forward
```

defers delivery if **/usr/forwards** appears not to exist. This can be useful if the directory is NFS mounted. If **check_local_user** is also set, **file_directory** takes precedence in determining the directory name for non-absolute files.

If **forwardfile** sets up a delivery to a file or a pipe command and the **home_directory** option is not set, then the directory specified by **file_directory**, or if that is not set, the home directory obtained from **check_local_user** is associated with the address during delivery.

file_transport

Type: *string*
Default: *unset*

A **forwardfile** director sets up a direct delivery to a file when a path name not ending in a slash is specified as a new ‘address’. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

filter

Type: *boolean*
Default: *false*

If this option is set, and the forward file starts with the text ‘# Exim filter’, then it is interpreted as a set of filtering commands instead of a list of forwarding addresses. Details of the syntax and semantics of filter files are described in a separate document entitled *Exim’s User interface to mail filtering*; this is intended for use by end users.

In addition to the commands described therein, there are some extra commands that are permitted only in system filter files, or if **allow_system_actions** is set. These are described in chapter 41.

The logging facility in filter files is available only if the filter is being run under some unprivileged uid. The system configuration must specify that **seteuid()** is available, either **user** or **check_local_user** must be set on the director, **forbid_filter_log** must not be set, and the global **security** setting must not be ‘setuid’. Writing the log takes place while the filter file is being interpreted, that is, at directing time. 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.

forbid_file

Type: *boolean*
Default: *false*

If this option is true, then this director may not generate a new address which specifies delivery to a local file. If it attempts to do so, a delivery failure occurs.

forbid_filter_logwrite

Type: *boolean*
Default: *false*

If this option is true, use of the logging facility in filter files is not permitted. This is in any case available only if the filter is being run under some unprivileged uid, which is normally the case for ordinary users’ **.forward** files on a system with **seteuid()** available.

forbid_include

Type: *boolean*
Default: *false*

If this option is true, then items of the form

`:include:`<*path name*>

are not permitted, and if one is encountered, the message is frozen.

forbid_pipe

Type: *boolean*
Default: *false*

If this option is true, then this director may not generate a new address which specifies delivery to a pipe. If it attempts to do so, a delivery failure occurs.

forbid_reply

Type: *boolean*
Default: *false*

If this option is true, then this director may not generate an automatic reply message. If it attempts to do so, a delivery failure occurs. Automatic replies can be generated only from filter files, not from traditional forward files.

freeze_missing_include

Type: *boolean*
Default: *true*

If a file named by the 'include' mechanism fails to open, delivery is frozen if this option is true. Otherwise, delivery is just deferred. Unsetting this option can be useful if included files are NFS mounted and may not always be available.

home_directory

Type: *string*
Default: *unset*

If this option is set, it associates a home directory with any address that **forwardfile** directs to a local transport because it specifies a file name or pipe command. The option string is expanded and set as the home directory during the delivery process (that is, at transport time), unless overridden by a setting on the transport. If **home_directory** is not set, then the directory specified by **file_directory**, or if that is not set, the home directory obtained from **check_local_user** is associated with the address during delivery. See chapter 13 for details of the local delivery environment. This option has no effect during the running of the **forwardfile** director.

In installations where users' **.forward** files are not kept in their home directories, both **check_local_user** and **file_directory** may be set, which leads to the **file_directory** value being used as the default. It is no good specifying

```
  home_directory = $home
```

because the same value is used for **$home**. A special string value is provided for use in this case. If **home_directory** is set to the string 'check_local_user' then the user's home directory path is set as the home directory during delivery.

ignore_eacces

Type: *boolean*
Default: *false*

If this option is set and an attempt to open the forward file yields the EACCES error (permission denied) then **forwardfile** behaves as if the file did not exist.

ignore_enotdir

Type: *boolean*
Default: *false*

If this option is set and an attempt to open the forward file yields the ENOTDIR error (something on the path is not a directory) then **forwardfile** behaves as if the file did not exist.

match_directory

Type: *string*
Default: *unset*

If this option is set with **check_local_user**, the user's home directory, as obtained from **getpwnam()**, must match the given string. If it does not, the director fails to match the address. The string is expanded before use if it contains any $ characters. If the expansion fails, Exim panics, unless the failure was explicitly triggered by a 'fail' item in a conditional sub-expression in the expansion, in which case the director just fails to handle the address.

If the expanded string starts with an asterisk, then the remainder must match the end of the home directory name; if it starts with a circumflex, a regular expression match is performed. In fact, the matching process is the same as is used for domain list items and may include file lookups.

modemask

Type: *octal integer*
Default: *022*

This specifies mode bits which must not be set for the forward file. If they are set, the director defers.

one_time

Type: *boolean*
Default: *false*

Sometimes the fact that Exim re-processes forward files each time it tries to deliver a message causes problems. This is particularly true in the case of mailing lists (see chapter 36).

If **one_time** is set and any addresses generated by the director fail to deliver at the first attempt, the failing addresses are added to the message as 'top level' addresses, and the parent address that generated them is marked 'delivered'. Thus expansion via the forward file does not happen again at the next delivery attempt. To ensure that **forwardfile** generates only addresses (as opposed to pipe or file deliveries or autoreplies) **forbid_file** and **forbid_pipe** must also be set, as must **forbid_reply** if **filter** is set.

The original top-level address is remembered with each of the generated addresses, and is output in any log messages. However, any intermediate parent addresses are not recorded. This makes a difference to the log only if **log_all_parents** is set. It is expected that **one_time** will typically be used for mailing lists, where there is normally just one level of expansion.

owners

Type: *string-list*
Default: *unset*

This specifies a list of permitted owners for the file. These are in addition to the local user in the case when **check_local_user** is set. If **owners** is unset and **check_local_user** is false, no check on the ownership is done. If the file is not correctly owned, the director fails and the message is frozen.

owngroups

Type: *string-list*
Default: *unset*

This specifies a list of permitted groups for the file. These are in addition to the local user's group in the case when **check_local_user** is set. However, a check on the group is made only when **check_group** is set. If the file's group is not correct, the director fails and the message is frozen.

pipe_transport

Type: *string*
Default: *unset*

A **forwardfile** director sets up a direct delivery to a pipe when a string starting with a vertical bar character is specified as a new ‘address’. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

qualify_preserve_domain

Type: *boolean*
Default: *false*

If this is set and an unqualified address (one without a domain) is generated, it is qualified with the domain of the incoming address instead of the global setting in **qualify_recipient**.

reply_transport

Type: *string*
Default: *unset*

A **forwardfile** director sets up a delivery to an **autoreply** transport when a **mail** or **vacation** command is used in a filter file. The transport used is specified by this option, which, after expansion, must be the name of a configured transport.

rewrite

Type: *boolean*
Default: *true*

If this option is set false, addresses generated by the director are not subject to address rewriting. Otherwise, they are treated like new addresses.

seteuid

Type: *boolean*
Default: *false*

This option may not be set unless the compile-time configuration in the OS-specific configuration files specifies that the **seteuid()** function is available in the operating system. In addition, either the **check_local_user** or the generic **user** and **group** options must be set. A configuration error occurs if these conditions do not hold.

When this option is true, the **seteuid()** and **setegid()** functions are called to change the effective uid and gid before accessing the home directory and the file. If both **check_local_user** and **user** are set, the uid is taken from the latter. If the generic **initgroups** option is set, **initgroups()** is called to initialise the group list with all the user’s groups. The user remains set during interpretation of a filter file; if it writes log entries the log file must be accessible to the uid or gid. Changing uid is necessary in two circumstances:

(i) When Exim is configured to change the effective uid from root to the Exim user (using **seteuid()**) while running the directors. See chapter 49 for details.

(ii) When users’ home directories are NFS mounted, and root access is not exported to the local host, to allow for cases when the files are not world-readable.

The **forwardfile** director can detect the first of these cases, and it always uses **seteuid()**, regardless of the setting of this option, since it does not make sense to do otherwise.

On a system without the **seteuid()** function, but with NFS home directories that do not export root, it is necessary for forward files to be world-readable.

skip_syntax_errors

Type: *boolean*
Default: *false*

If **skip_syntax_errors** is set for a non-filter forward file, a malformed address that causes a parsing error is skipped, and an entry is written to the main log. This may be useful for mailing lists that are automatically managed, but note the inherent danger. If all the addresses in the list are malformed, the original address is passed on to subsequent directors.

If **skip_syntax_errors** is set for an Exim filter file, any syntax error in the filter file causes filtering to be abandoned, the incident is logged, and the address is passed on to the next director.

If **skip_syntax_errors** is set for the director that handles users' **.forward** files, it should normally be done in conjunction with

```
  syntax_errors_to = $local_part@domain
```

in order to pass the error report back to the owner of the **.forward** file. Additional locally-written explanation can be included by setting **syntax_errors_text**.

syntax_errors_text

Type: *string*
Default: *unset*

See **syntax_errors_to**.

syntax_errors_to

Type: *string*
Default: *unset*

This option applies only when **skip_syntax_errors** is set. If any addresses are skipped because of syntax errors, a mail message is sent to the address specified by **syntax_errors_to**, giving details of the failing address(es). If **syntax_errors_text** is set, its contents are expanded and placed at the head of the error message. Often it will be appropriate to set **syntax_errors_to** to be the same address as **errors_to**.

23. The localuser director

The **localuser** director checks whether the local part of an address is the login of a local user, by calling the **getpwnam()** function. If it is, and if other conditions set by options are met, it accepts the address and sets up a transport for it. The user's uid, gid, and home directory are set up by default to be used while running the delivery process. The generic **transport** option must always be specified, unless the generic **verify_only** option is set.

The generic **require_files** option may contain references to **$home** when used with this director. Thus it is possible to pick out all users with particular files in their home directories and route their mail to a specific transport. This could be used, for example, to check for a **.procmailrc** file and then to route delivery via **procmail** if one is found.

current_directory

Type:	*string*
Default:	*unset*

This option string is expanded and set as the current directory during the delivery process, unless overridden by a setting on the transport. See chapter 13 for details of the local delivery environment.

home_directory

Type:	*string*
Default:	*unset*

This option overrides the home directory that is obtained from the **getpwnam()** function. The string is expanded and set as the home directory during the delivery process, unless overridden by a setting on the transport. See chapter 13 for details of the local delivery environment.

When processing the **require_files** generic option, the value of **$home** is the value of this option if it is set. Otherwise the value of **$home** is the user's home directory as obtained from **getpwnam()**.

match_directory

Type:	*string*
Default:	*unset*

If this option is set, the user's home directory, as obtained from **getpwnam()**, must match the given string. If it does not, the director fails to match the address. This provides a way of partitioning the local users by home directory. The string is expanded before use if it contains any $ characters. If the expansion fails, Exim panics, unless the failure was explicitly triggered by a 'fail' item in a conditional sub-expression in the expansion, in which case the director just fails to handle the address.

If the expanded string starts with an asterisk, then the remainder must match the end of the home directory name; if it starts with a circumflex, a regular expression match is performed. In fact, the matching process is the same as is used for domain list items and may include file lookups.

On central systems at Cambridge, when a user account is cancelled, it remains in the password file for a while, with the home directory set to **/home/CANCELLED**. We use the **match_directory** option to detect mail addressed to such users and bounce it with an explanatory message.

24. The smartuser director

The **smartuser** director matches *any* local part, so it can be used to handle local addresses that all other directors have failed. It is, of course, subject to the generic director options, so specific instances can be used for all addresses in certain domains, or all local parts with certain prefixes or suffixes, or specific local parts, or any other generic condition.

Smartuser can generate a new address from the old one, and cause that to be reprocessed, or it can set a transport, optionally changing the address. Common uses are to pipe the message to a script that generates an information message to be returned to the sender, or to send the message to another host for processing.

new_address

Type: *string*
Default: *unset*

This option specifies a new address, to replace the current one. It must be a qualified address (that is, contain an @ character). The string is expanded, so settings such as

```
new_address = ${local_part}@some.new.host
```

can be used, or a file lookup on the local part can be done. If the expansion fails as a result of an explicit 'fail' item in an expansion sub-expression, the director just fails to handle the address. Otherwise, an expansion failure is treated as a serious configuration error, and causes a panic, unless this **panic_expansion_fail** is set false, in which case the same action is taken as for 'fail'.

The new address is rewritten by Exim's normal rewriting rules (see chapter 32) unless the **rewrite** option is turned off.

If no transport is specified, **new_address** is required, and the new address is processed by the directors and routers in the normal way, as if it were the result of aliasing or forwarding. In particular, if it is a duplicate of any other address in the message, it is discarded.

On the other hand, if a transport is specified for **smartuser**, the new address replaces the old one when the message is delivered by the given transport, and no checking for duplication takes place. The original address is available to the transport via the expansion variables **$original_local_part** and **$original_domain**.

panic_expansion_fail

Type: *boolean*
Default: *true*

See **new_address** above.

rewrite

Type: *boolean*
Default: *true*

If this option is set false, the address specified by **new_address** is not subject to rewriting.

25. Additional generic options for routers

The following additional generic options apply to all routers, in addition to the common generic options for both directors and routers which are described in chapter 19. Routers are concerned with addresses whose domains do not match something in **local_domains**.

pass_on_timeout

Type:	*boolean*
Default:	*false*

If a router times out during a host lookup, it normally causes deferral of the address. If **pass_on_timeout** is set, the address is instead passed on to the next router. This may be helpful for systems that are intermittently connected to the Internet, or those that want to pass to a smart host any messages that cannot immediately be delivered.

There are occasional other temporary errors that can occur while doing DNS lookups. They are treated in the same way as a timeout, and this option applies to all of them.

self

Type:	*string*
Default:	*"freeze"*

This option specifies what is to happen if routing a remote address ends up pointing at the local host, or at a host whose name matches **hosts_treat_as_local**. Normally this indicates either an error in Exim’s configuration (for example, the domain should be listed as local), or an error in the DNS (for example, the MX shouldn’t point at this host). However, this situation is not confined to the use of MX records, and the **self** option can be used on any router .

The default action is to freeze the message. The following alternatives are provided for use in special cases:

- **defer**
  Delivery of the message is tried again later.

- **reroute:** <*domain*>
  The domain is changed to the given domain, and the address is passed back to be reprocessed by the directors and routers. No rewriting of headers takes place.

- **reroute: rewrite:** <*domain*>
  The domain is changed to the given domain, and the address is passed back to be reprocessed by the directors and routers. Any headers that contain the original domain are rewritten.

- **local**
  The address is passed to the directors, as if its domain were a local domain, even though it does not match anything in **local_domains**. This can be used to treat all domains whose lowest MX records point to the host as local domains. During subsequent directing and delivery the variable **$self_hostname** is set to the name of the local host that the router encountered. This can be used to distinguish between different cases for hosts with multiple names.

- **fail_soft**
  The router fails, passing the address to the following router. During subsequent routing and delivery, the variable **$self_hostname** contains the name of the local host that the router encountered. This can be used to distinguish between different cases for hosts with multiple names. This setting overrides a setting of **no_more** on the router, so a combination of **fail_soft** and **no_more** ensures that only those addresses that routed to the local host are passed on. Without **no_more**, failing addresses would also be passed to the next router.

- **fail_hard**
  The router fails, and the address is not passed to any following routers. Consequently, delivery fails and an error report is generated.
- **send**
  The anomaly is ignored and the message is transmitted anyway. This setting should be used with extreme caution. It makes sense only in cases where the program that is listening on the SMTP port is not this version of Exim. That is, it must be some other MTA, or Exim with a different configuration file that handles the domain in another way.

When a router just rewrites, that is, does not set up IP addresses, the **self** option is not relevant.

26. The domainlist router

The **domainlist** router compares a list of domain patterns with the domain it is trying to route. When a match is found, the information associated with the pattern can specify several different actions:

- The message can be sent to a specific host, or one of a number of hosts.
- The domain name can be replaced by a new name, which can be

  (i) passed to the next router; or

  (ii) looked up directly in the DNS, with or without MX processing; or

  (iii) looked up using **gethostbyname()**.

  Of course, **gethostbyname()** may well do its own DNS lookup, but it does not do MX processing, and it may also reference other sources of information, such as **/etc/hosts**. When Exim is compiled with IPv6 support, if a host that is looked up in the DNS has both A and AAAA records, all the addresses are used. See **README.IPV6** for general information about IPv6 support.

The list of patterns can be specified as an option string, or looked up in a file or database, or both; at least one of **route_list**, **route_file**, **route_query**, or **route_queries** must be set. A transport must be set when the routing is completed by this router, that is, when the address is not passed on to subsequent routers, unless **verify_only** is set. Each routing entry can specify its own transport, with the generic **transport** option acting as a default for those that don't.

host_find_failed

Type:	*string*
Default:	*"freeze"*

This option controls what happens if a host which **domainlist** tries to look up because an address has been specifically routed to it does not exist. The option can be set to one of

```
freeze
defer
fail_soft
fail_hard
```

The default assumes that this state is a serious configuration error. The difference between 'fail_soft' and 'fail_hard' is that the former causes the address to be passed to the next router, overriding **no_more**, while the latter does not, causing the address to fail completely. This option applies only to a definite 'does not exist' state; if a host lookup gets a temporary error, delivery is deferred unless the generic **pass_on_timeout** option is set.

modemask

Type:	*octal integer*
Default:	*022*

This specifies mode bits which must not be set for the route file. If they are set, the director fails and the message is frozen.

owners

Type:	*string-list*
Default:	*unset*

This specifies a list of permitted owners for the route file. If it is unset, no check on the ownership is done. If the file is not owned by a user in the list, the router fails and the message is frozen.

owngroups

Type: *string-list*
Default: *unset*

This specifies a list of permitted groups for the route file. If it is unset, no check on the file's group is done. If the file's group is not in the list, the router fails and the message is frozen.

qualify_single

Type: *boolean*
Default: *true*

For any domain that is looked up in the DNS, the resolver option that causes it to qualify single-component names with the default domain (RES_DEFNAMES) is set. For example, on a machine called **dictionary.ref.book**, looking up the domain **thesaurus** would cause the name **thesaurus.ref.book** to be looked up.

route_file

Type: *string*
Default: *unset*

If this option is set, **search_type** must be set to one of the single-key lookup types, and **route_query** must not be set. See chapter 6 for details of file and database lookups. The domain being routed is used as the key for the lookup, and the resulting data must be a list of routing rules in the form described below. The file name is expanded before use.

route_list

Type: *string-list, semicolon-separated*
Default: *unset*

This string is a list of routing rules, in the form defined below. Note that, unlike most string lists, the items are separated by semicolons. This is so that they may contain colon-separated host lists.

route_queries

Type: *string*
Default: *unset*

This option is an alternative to **route_query**; the two options are mutually exclusive. The difference is that **route_queries** contains a colon-separated list of queries, which are tried in order until one succeeds or defers, or all fail. Any colon characters actually required in an individual query must be doubled, in order that they not be treated as query separators.

route_query

Type: *string*
Default: *unset*

If this option is set, **search_type** must be set to a query-style lookup type, and **route_file** must not be set. See chapter 6 for details of file and database lookups. The query is expanded before use, and the expansion variable **$domain** contains the domain being routed. The data returned from the lookup must be a list of routing rules, in the form described below.

search_parents

Type: *boolean*
Default: *false*

For any domain that is looked up in the DNS, the resolver option that causes it to search parent domains (RES_DNSRCH) is set if this option is true. This is different from the **qualify_single** option in that it applies to domains containing dots. For example, on a machine in the **fict.book** domain, when looking up **teaparty.wonderland** initially fails, the resolver automatically tries **teaparty.wonderland.fict.book** if this option is set.

search_type

Type: *string*
Default: *unset*

This option is mandatory when **route_file**, **route_query**, or **route_queries** is specified. It must be set to one of the supported search types (for example, **lsearch**). See chapter 6.

For single-file lookups, the name may be preceded by ‘partial-’, indicating a simple wildcard file lookup that works as follows:

(a) Exim first tries to look up the domain exactly as given.

(b) If that fails, it adds ‘∗.’ on the front of the domain, and looks that up.

(c) If that fails, it replaces the first component of the domain with ‘∗’ and tries that, and continues chopping off components in this way until either the lookup succeeds, or there are fewer than two non-∗ components left.

Thus, for example, if you put an entry keyed by **∗.austen.fict.film** in your database, that entry will be used for

(1) **austen.fict.film** by rule (b) above, having failed on rule (a). (If you are worried about the resource waste implied by this, you can always add an entry for **austen.fict.film** as well.)

(2) **emma.austen.fict.film** at the first attempt in rule (c), having failed on rules (a) and (b).

A domain such as **jane.fict.film** will fail, having tried 3 lookups: **jane.fict.film**, **∗.jane.fict.film**, **∗.fict.film**, but it won’t waste effort looking up **∗.film** because that has only one non-∗ component. In fact, the minimum number of components can be altered by including a number immediately before the hyphen. For example, ‘partial4-dbm’ specifies a minimum of four non-∗ components.

26.1 Routing rules

Routing rules specified in **route_list** are scanned before **route_file**, **route_query** or **route_queries** are used. The contents of **route_list** is a string consisting of a sequence of routing rules, separated by semicolons. If a semicolon is needed in a rule, it can be entered as two semicolons. Empty rules are ignored. The format of each rule is

<*domain pattern*> <*host-list*> <*options*>

The following example contains a simple domain pattern and just one rule:

```
route_list = "dict.ref.book mail-1.ref.book:mail-2.ref.book byname"
```

The three parts of a rule are separated by white space. Each rule in a **route_list** must start with a single domain pattern, which is the only mandatory item in the rule. The pattern is in the same format as one item in a domain list (see section 7.12), that is, it may be wildcarded or a regular expression, or a file or database lookup (with semicolons doubled, because of the use of semicolon as a separator in a **route_list**). The rules in **route_list** are searched in order until one of the patterns matches the domain that is being routed. The host list and options are then used as described below.

If no rule in **route_list** matches the domain, it is used as the key for a lookup of the type specified by **search_type**, using **route_file**, **route_query**, or **route_queries**, as appropriate. The data returned from a successful lookup must be a string containing a host list and options, separated by white space. For example, a line in a linearly searched route file might be:

```
dict.ref.book:  mail-1.ref.book:mail-2.ref.book  byname
```

Note that there are two different uses of the colon character in this line. The first one is the delimiter of the key in the file, while the second is the normal list delimiter in the host list, which in this example consists of two host names. As both the host list and the options are not compulsory in a rule, the data returned from a lookup can legitimately be an empty string in some circumstances (see *Application of routing rules* below).

If the domain does not match anything in **route_list** and looking it up using **route_file**, **route_query** or **route_queries** also fails, then the router cannot handle the address, and it gets passed on to the next router, unless **no_more** is set.

26.2 Host list format

If a host list is present in the rule which matches the domain, it is expanded before use. If the pattern that matched the domain was a lookup item, the data that was looked up is available in the expansion variable **$value**.

The result of the expansion must be a colon-separated list of host names and/or IP addresses. Some string expansion items may contain white space, and if this is the case, the host list must be enclosed in single or double quotes, because otherwise white space terminates it. The numeric expansion variables are available during host list expansion. These are mainly used when the domain is matched against a regular expression domain pattern in a **route_list** string, but **$1** is also set when partial matching is done in a file lookup, and **$0** is always set to the entire domain.

The value of **$domain** is the original domain for the address. This may differ from **$0** if the address has been processed by a previous **domainlist** router which passed on a different routing domain.

If the expansion of the host list is forced to fail (by using the ‘fail’ item in a conditional construction), the router just fails to handle the address, and (unless **no_more** is set) it gets passed on to the next router. If expansion fails for some other reason, the message is frozen, since this is considered to be a configuration error.

26.3 Options format

Options can be present only if there is a host list. They are a sequence of words, but in practice no more than two are ever present. One of the words can be the name of one of the configured transports, and this overrides the **transport** option on the router for this particular routing rule only. The other word (if present) specifies how the IP addresses of the hosts in the host list are to be found:

- **byname**: use **gethostbyname()**, or use literal IP addresses if present. Literal IP addresses are written without any surrounding square brackets.
- **bydns**: use the DNS, doing the full MX and A record processing.
- **bydns_a**: look up A records for the host(s) in the DNS; fail if there are none.
- **bydns_mx**: look up MX records for the host(s) in the DNS; fail if there are none.

The **qualify_single** and **search_parents** options apply to any DNS lookups that are done. If no IP address for a host can be found, what happens is controlled by the **host_find_failed** option.

26.4 Application of routing rules

When a rule has been found that matches the current domain, either by matching one of the rules in **route_list**, or by a successful lookup in **route_file** or using **route_query** or **route_queries**, the host list and options are used in a number of different ways, depending on which are present and on whether a transport has been specified.

- If there is no host list (and therefore necessarily no options either), then a local transport (that is, not an SMTP transport) must be specified for the router via the generic **transport** option, unless the driver is being used only for verification (**verify_only** is set). In this case, if there is no transport and no host list, the address is taken as verified. Otherwise, failure to specify a local transport in the absence of a host list is a configuration error. The address is routed to the transport. In all other cases, a host list must be provided.
- If there is a host list, and a local transport is specified either by the generic **transport** option, or by an option item in the rule, then the host list must contain just a single host name which is passed to the transport in the **$host** variable. Any **by***xxx* options are ignored.

- If no **by***xxx* option is present, then any remote transport setting is ignored, and there must be just one name in the host list. The address is passed on to the next router, with the domain being routed being replaced by the name from the host list. However if the expansion variable **$domain** is used in any subsequent router, it still refers to the original domain.
- Otherwise, a remote (that is, SMTP) transport must be specified, unless the driver is being used only for verification (**verify_only** is set). The transport is specified either via the generic **transport** option or by a transport name as an option setting, and there may be many hosts in the list. Their IP addresses are looked up according to the **by***xxx* option. If any of them are found to be the local host, that one and all those that follow it are discarded. If the first host is found to be the local host, then the generic **self** option specifies what happens. Otherwise, the address is passed to the specified transport, along with the ordered list of hosts. The transport will try delivering to each host in turn, until one accepts the message.

The various different possibilities for configuring the **domainlist** router make it possible to use it for a number of different routing requirements, as shown in the examples in the next section.

26.5 Domainlist examples

In some of the examples that follow, the presence of the **remote_smtp** transport, as defined in the default configuration file, is assumed.

- Routing to a gateway to another mail environment can be set up using a wildcarded domain pattern that matches some pseudo top-level domain. For example, to route certain addresses to UUCP and Bitnet gateways:

```
uucp_bitnet:
  driver = domainlist
  route_list = "*.uucp   uugateway.fict.book; \
                *.bitnet bngateway.ref.book"
```

  The two rules match domains ending in **.uucp** and **.bitnet** respectively, and because no options or transport are specified in either case, the name of the appropriate gateway domain is taken from the host list and passed to subsequent routers for further routing. So, for example, mail addressed to **user@faraway.uucp** is routed by applying subsequent routers to the domain **uugateway.fict.book** to determine where to send it.

  If there are two hosts servicing one of these domains and they are not connected to a single domain name (by MX records for example), you may want to quote two names in the host list portion of a rule. In this case, you have to specify one of the **by***xxx* options, to get the names looked up by **domainlist**, since it can pass on only a single domain name to other routers. A transport must also be provided:

```
uucp:
  driver = domainlist
  transport = remote_smtp
  route_list = "\
    *.uucp uugate1.fict.book:uugate2.fict.book byname"
```

  In this case, no further routers are called.

- A host that is itself a gateway can ‘deliver’ messages to pipes or into files in batched SMTP format for onward transportation by some other means. In this case, the route list entry can be as simple as a single domain name in a configuration like this:

```
route_append:
  driver = domainlist
  transport = batchsmtp_appendfile
  route_list = gated.domain
```

though often a pattern is used to pick up more than one domain. If there are several domains or groups of domains with different transport requirements, different transports can be listed in the routing information:

```
route_append:
  driver = domainlist
  route_list = "\
    *.gated.domain1  $domain  batch_appendfile; \
    *.gated.domain2  \
      ${lookup{$domain}dbm{/domain2/hosts}{$value}fail} \
      batch_pipe"
```

The first of these just passes the domain in the **$host** variable, which doesn't achieve much (since it is also in **$domain**) but the second does a file lookup to find a value to pass, causing the router to fail to handle the address if the lookup fails.

- Routing mail directly to UUCP software is a specific case of the use of **domainlist** in a gateway to another mail environment. This is an example of one way it can be done, taken from a real configuration:

```
# Transport
uucp:
  driver = pipe
  user = nobody
  command = "/usr/local/bin/uux -r - \
    ${substr_-5:$host}!rmail ${local_part}"
  return_fail_output = true

# Router
uucphost:
  transport = uucp
  driver = domainlist
  route_file = /usr/local/exim/uucphosts
  search_type = lsearch
```

The file **/usr/local/exim/uucphosts** contains entries like

```
darksite.ethereal.ru:           darksite.UUCP
```

It can be set up more simply without adding and removing '.UUCP' but this way makes clear the distinction between the domain name **darksite.ethereal.ru** and the UUCP host name **darksite**.

- A *mail hub* is a machine which receives mail for a number of domains via MX records in the DNS and delivers it via its own private routing mechanism. Often the final destinations are behind a firewall, with the mail hub being the one machine that can connect to machines both inside and outside the firewall. The **domainlist** router can be set up for this kind of purpose:

```
through_firewall:
  driver = domainlist
  transport = remote_smtp
  route_file = /internal/host/routes
  search_type = lsearch
```

For a small number of cases, the routing could be inline, using the **route_list** option, but for a larger number a file lookup would be easier to manage, and the file containing the internal routing might contain lines like this:

```
dict.ref.book:  mail-1.ref.book:mail-2.ref.book  byname
```

The DNS would be set up with an MX record for **dict.ref.book** pointing to the mail hub, which would then then forward mail for **dict.ref.book** to one of the two specified machines, looking up their addresses using **gethostbyname()**.

If the domain names are in fact the names of the machines to which the mail is to be sent by the mail hub, then the configuration can be quite simple. For example,

```
hub_route:
  driver = domainlist
  transport = remote_smtp
  route_list = *.rhodes.tvs  $domain  byname
```

This configuration routes domains that match ***.rhodes.tvs** by calling **gethostbyname()** on the domain that matched. A similar approach can be taken if the host name can be obtained from the domain name by simple manipulation that the expansion facilities can handle.

- The **domainlist** router can also be used to forward all non-local mail to a *smart host* by using a configuration like

  ```
  smart_route:
    driver = domainlist
    transport = remote_smtp
    route_list = "*  smarthost.ref.book  bydns_a"
  ```

  which causes all messages containing remote addresses to be sent to the single host **smarthost.ref.book**, whose address (in this example) is obtained from its DNS address record. If a colon-separated list of smart hosts is given, they are tried in order. A router like this should be the last one in the configuration file, since it will route any domain whatsoever.

- A **domainlist** router can be used to force success or failure on verification of remote addresses by setting **verify_only** (and **verify_sender** or **verify_recipient** if required). If failure is wanted, set **fail_verify**. No transports or hosts need be defined.

27. The ipliteral router

This router succeeds if the ‘domain’ being routed takes the form of an RFC 822 domain literal, that is, an IP address in dotted-quad notation enclosed in square brackets. For example, this router handles the address

```
  root@[111.1.1.1]
```

by setting up delivery to the host with that IP address. If an IP literal turns out to refer to the local host, the generic **self** option determines what happens. The RFCs require support for domain literals, though it seems anachronistic in today’s Internet. There are no private options for this router; a transport must be set using the generic **transport** option.

28. The iplookup router

The **iplookup** router was written to fulfil a specific requirement in Cambridge. For this reason, it is not included in the binary of Exim by default. If you want to include it, then you must set

```
ROUTER_IPLOOKUP=yes
```

in your **Local/Makefile** configuration file.

The **iplookup** router routes an address by sending it over a TCP or UDP connection to one or more specific hosts. The host can then return the same or a different address – in effect rewriting the recipient address in the message's envelope. If this process fails, the address can be passed on to other routers, or delivery can be deferred.

Background, for those that are interested: We have an Oracle database of all Cambridge users, and one of the bits of data it maintains for each user is where to send mail addressed to <*user*>**@cam.ac.uk**. The MX records for **cam.ac.uk** point to a central machine that has a large alias list that is abstracted from the database. Mail from outside is switched by this system, and originally internal mail was also done this way. However, this resulted in a fair number of messages travelling from some of our larger systems to the switch and back again. The Oracle machine now runs a UDP service that can be called by the **iplookup** router in Exim to find out where <*user*>**@cam.ac.uk** addresses really have to go; this saves passing through the central switch, and in many cases saves doing any remote delivery at all.

Since **iplookup** is just a rewriting router, a transport must *not* be specified for it.

hosts

Type:	*string*
Default:	*unset*

This option must be supplied. Its value is a colon-separated list of host names. The hosts are looked up using **gethostbyname()** and are tried in order until one responds to the query.

optional

Type:	*boolean*
Default:	*false*

If **optional** is true, then if no response is obtained from any host, the address is passed on to the next router. If **optional** is false, delivery to this address is deferred.

port

Type:	*integer*
Default:	*0*

This option must be supplied. It specifies the port number for the TCP or UDP call.

protocol

Type:	*string*
Default:	*udp*

This option can be set to 'udp' or 'tcp' to specify which of the two protocols is to be used.

query

Type:	*string*
Default:	*"${local_part}@${domain} ${local_part}@${domain}"*

This defines the content of the query that is sent to the remote hosts. The repetition serves as a way of checking that a response is to the correct query in the default case (see **response_pattern** below).

reroute

Type: *string*
Default: *unset*

If this option is not set, the rerouted address is precisely the byte string returned by the remote host, up to the first white space, if any. If set, the string is expanded to form the rerouted address. It can include parts matched in the response by **response_pattern** by means of numeric variables such as **$1**, **$2**, etc. The variable **$0** refers to the entire input string, whether or not a pattern is in use. In all cases, the rerouted address must end up in the form <*local_part*>@<*domain*>.

response_pattern

Type: *string*
Default: *unset*

This option can be set to a regular expression that is applied to the string returned from the remote host. If the pattern does not match the response, the router fails. If **response_pattern** is not set, no checking of the response is done, unless the query was defaulted, in which case there is a check that the text returned after the first white space is the original address. This checks that the answer that has been received is in response to the correct question. For example, if the response is just a new domain, the following could be used:

```
response_pattern = "^([^@]+)$"
reroute = "${local_part}@${1}"
```

service

Type: *integer*
Default: *0*

This is an alternative name for the **port** option.

timeout

Type: *time*
Default: *5s*

This specifies the amount of time to wait for a response from the remote machine. The same timeout is used for the **connect()** function for a TCP call. It does not apply to UDP.

29. The lookuphost router

The **lookuphost** router looks up the hosts that handle mail for the given domain either via the **gethostbyname()** function, or by using the DNS directly. A transport must always be set for this router, unless **verify_only** is set.

When the DNS is used, MX records are looked up first, followed by A records if no MX records are found, unless the domain matches **mx_domains**. Unless they have the highest priority (lowest MX value), MX records that point to the local host, or to any host name that matches **hosts_treat_as_local**, are discarded, together with any other MX records of equal or lower priority.

If the host pointed to by the highest priority MX record or the host looked up by **gethostbyname()** is the local host, or matches **hosts_treat_as_local**, then what happens is controlled by the generic **self** option.

check_secondary_mx

Type:	*boolean*
Default:	*false*

If this option is set, the router fails unless the local host is found in (and removed from) the list of hosts obtained by MX lookup. This can be used to process domains for which the local host is a secondary mail exchanger differently to other domains.

gethostbyname

Type:	*boolean*
Default:	*false*

If this is true, the **gethostbyname()** function is used and the options relating to the DNS are ignored. Otherwise, the name is looked up directly in the DNS. Of course, **gethostbyname()** may do its own DNS lookup for an A record (no MX processing is involved), but it may also access other sources of information such as **/etc/hosts**.

When Exim is compiled with IPv6 support, if a host that is looked up in the DNS has both A and AAAA records, all the addresses are used. See **README.IPV6** for general information about IPv6 support.

mx_domains

Type:	*domain-list*
Default:	*unset*

This option applies to domains that are looked up directly in the DNS (**gethostbyname** is not set) for non-source-routed RFC 822 addresses (that is, addresses that do not start with @). A domain which matches **mx_domains** is required to have an MX record in order to be recognised. For example, if all the mail hosts in **fict.book** are known to have MX records, except for those in **discworld.fict.book**, options of the form

```
mx_domains = ! *.discworld.fict.book : *.fict.book
```

could be used. This would cause messages addressed to a machine that matched the option but had only an A record to be bounced immediately instead of sitting on the queue until the delivery timed out. Note, however, that for source-routed RFC 822 addresses (ones that start with @) this restriction does not apply, as the first domain in such an address is a machine name. The **collapse_source_routes** main configuration option provides a way of locking out the use of source routes.

qualify_single

Type: *boolean*
Default: *true*

If domains are being looked up in the DNS (**gethostbyname** is false), then the resolver option that causes it to qualify single-component names with the default domain (RES_DEFNAMES) is set. For example, on a machine called **dictionary.ref.book**, looking up the domain **thesaurus** would cause the name **thesaurus.ref.book** to be looked up internally in the resolver. Exim itself still looks up the single name.

rewrite_headers

Type: *boolean*
Default: *true*

An abbreviated name may be expanded to its full form by both **gethostbyname()** or by DNS lookup, or as a result of the **widen_domains** option. For example, if an address is specified as **dormouse@teaparty**, the domain might get expanded to **teaparty.wonderland.fict.book**. If this option is true, then all occurrences of the abbreviated name in the headers of the message are rewritten with the full name. This option should be turned off only when it is known that no message is ever going to be sent outside an environment where the abbreviation makes sense.

When an MX record is looked up in the DNS and matches a wildcard record, nameservers normally return a record containing the name that has been looked up, making it impossible to detect whether a wildcard was present or not. However, some nameservers have recently been seen to return the wildcard entry. If the name returned by a DNS lookup begins with an asterisk, it is not used for header rewriting.

search_parents

Type: *boolean*
Default: *false*

If domains are being looked up in the DNS (**gethostbyname** is false), then the resolver option that causes it to search parent domains (RES_DNSRCH) is set if this option is true. This is different from the **qualify_single** option in that it applies to domains containing dots. For example, on a machine in the **fict.book** domain, when looking up **teaparty.wonderland** initially fails, the resolver automatically tries **teaparty.wonderland.fict.book** if this option is set. The default setting of this option used to be true, but this causes problems in domains that have a wildcard MX record, because any domain that does not have its own MX record then matches the local wildcard. The default was changed to false in Exim 1.80.

widen_domains

Type: *string-list*
Default: *unset*

If a lookup fails and this option is set, each of its strings in turn is added onto the end of the domain, and the lookup is tried again. For example, if

```
widen_domains = "fict.book:ref.book"
```

is set and a lookup of **klingon.dictionary** fails, then **klingon.dictionary.fict.book** is looked up, and if this fails, then **klingon.dictionary.ref.book** is tried. This option applies to lookups using **gethostbyname()** as well as to DNS lookups. Note that when the DNS is being used for lookups, the **qualify_single** and **search_parents** options cause some widening to be undertaken inside the DNS resolver.

30. The queryprogram router

The **queryprogram** router routes a domain by running an external command and acting on its output. This is an expensive way to route, and is intended mainly for use in lightly-loaded systems, or for performing experiments. However, if it is possible to use the **domains**, **local_parts** or **condition** generic options to skip this router for most addresses, then it could sensibly be used in special cases. There are the following private options:

command

Type:	*string*
Default:	*unset*

This option must be set, and must start with a slash character. It specifies the command that is to be run. It is expanded before use. Failure to expand causes the router to fail and the message to be frozen.

command_group

Type:	*string*
Default:	*unset*

This option specifies a gid to be set when running the command. If it begins with a digit it is interpreted as the numerical value of the gid. Otherwise it is looked up using **getgrnam()**.

command_user

Type:	*string*
Default:	*unset*

This option specifies the uid which is set when running the command. If it begins with a digit it is interpreted as the numerical value of the uid. Otherwise, it is looked up using **getpwnam()** to obtain a value for the uid and, if **command_group** is not set, a value for the gid also.

current_directory

Type:	*string*
Default:	*unset*

This option specifies an absolute path which is made the current directory before running the command. If it is not set, '/' is used.

timeout

Type:	*time*
Default:	*1h*

If the command does not complete within the timeout period, its process group is killed and the message gets frozen. A value of zero time specifies no timeout.

If **command_user** is not specified, the command is run as 'nobody'. If the main configuration has not defined a user and group for 'nobody', then it is looked up using **getpwnam()**. If this fails, the router fails and the message is frozen.

In previous versions of Exim the **command_group** and **command_user** options were called **group** and **user**. Their names were changed when **group** and **user** became generic router options.

The standard output of the command is connected to a pipe, which is read when the command terminates. It should consist of a single line of output, containing up to five fields, separated by white space. The first field is one of the following words:

- OK: routing succeeded; the remaining fields specify what to do.

- FAIL: routing failed; pass the address to the next router.
- FORCEFAIL: routing failed; do not pass the address to any more routers.
- DEFER: routing could not be completed at this time; try again later.
- ERROR: some disastrous error occurred; freeze the message.

When the first word is not OK, the remainder of the line is an error message explaining what went wrong. For example:

```
FAIL  queryprogram cannot route to unseen.discworld.fict.book
```

Otherwise, the line must be formatted as follows:

`OK` <*transport name*> <*new domain*> <*option*> <*arbitrary text*>

The second field is the name of a transport instance, or a plus character, which means that the transport specified for the router using the generic **transport** option is to be used, if set.

If the third field is not empty or a single plus character, it is a new domain name to replace the current one. If a transport is specified and the fourth field is not empty or a plus character, it specifies the method of looking up the new name. This can be one of the words 'byname', 'bydns', 'bydns_a', or 'bydns_mx'. For example,

```
OK  smtp  gate.star.fict.book  bydns_a
```

causes the message to be sent using the **smtp** transport to the host **gate.star.fict.book**, whose address is looked up as a DNS address record. If the host turns out to be the local host, what happens is controlled by the generic **self** option.

The fifth field, if present, is made available to the transport via the expansion variable **$route_option**. For example, a line such as

```
OK special + + /computed/filename
```

sends the message to the **special** transport, which can use **$route_option** in its configuration to access the text '/computed/filename'.

The fourth and fifth fields are ignored and the new domain name (if any) is passed to the next router if no transport is specified in the response line (that is, a plus character is given) and the generic **transport** option is also unset.

31. Retry configuration

The fifth part of the configuration file contains a list of retry rules which control how often Exim tries to deliver messages that cannot be delivered at the first attempt. If there are no retry rules, Exim gives up after the first failure. The **-brt** command line option can be used to test which retry rule will be used for a given address or domain.

The most common cause of retries is temporary failure to deliver to a remote host. Exim's retry processing in this case is applied on a per-host (strictly, per IP address) basis, not on a per-message basis. Thus, if one message has recently been delayed, a new message to the same host does not immediately get tried, but waits for the host's retry time to arrive. If the value of **log_level** is greater than 4, the message 'retry time not reached' is written to the main log whenever a delivery is skipped for this reason. Section 42.2 contains more details of the handling of errors during remote deliveries.

Retry processing applies to directing and routing as well as to delivering, except as covered in the next paragraph. The retry rules do not distinguish between these three actions, so it is not possible, for example, to specify different behaviour for failures to route the domain **snark.fict.book** and failures to deliver to the host **snark.fict.book**. I didn't think anyone would ever need this added complication, so did not implement it. However, although they share the same retry rule, the actual retry times for routing, directing, and transporting a given domain are maintained independently.

When a delivery is not part of a queue run (typically an immediate delivery on receipt of a message), the directors are always run for local addresses, and local deliveries are always attempted, even if retry times are set for them. This makes for better behaviour if one particular message is causing problems (for example, causing quota overflow, or provoking an error in a filter file). If such a delivery suffers a temporary failure, the retry data gets updated as normal, and subsequent delivery attempts from queue runs occur only when the retry time for the local address is reached.

31.1 Retry rules

Each retry rule occupies one line and consists of three parts, separated by white space: a pattern, an error name, and a list of retry parameters. The rules are searched in order until one is found whose pattern matches the failing host or address.

The pattern may be a complete address (**local_part@domain**), a plain domain, a wildcarded domain (that is, starting with an asterisk), a domain lookup (as in a domain list), or a regular expression. The first form must be used with local domains only; in this case the local part may begin with an asterisk.

After a directing or local delivery failure, regular expressions and patterns containing local parts are normally matched against the complete address (**local_part@domain**). However, if there is no local part in a pattern that is not a regular expression, then the local part of the address isn't used in the matching. Thus an entry such as

```
lookingglass.fict.book         *  F,24h,30m;
```

matches any address whose domain is **lookingglass.fict.book**, whether this is a local or a remote domain, whereas

```
alice@lookingglass.fict.book   *  F,24h,30m;
```

can be used only if **lookingglass.fict.book** is a local domain. It applies to temporary failures involving the local part **alice**, but not to any other local parts.

If a local delivery is being used to collect messages for onward transmission by some other means (for example, as batched SMTP), a temporary failure may not be dependent on the local part at all. Both the **appendfile** and **pipe** transports have an option called **retry_use_local_part** which can be set false in order to suppress the inclusion of local parts when matching retry patterns for those transport instances. When this option is set, patterns containing local parts are skipped, and regular expressions are matched against the domain only.

For remote domains, when looking for a retry rule after a routing attempt has failed (for example, after a DNS timeout), each line in the retry configuration is tested only against the domain in the address. However, when looking for a retry rule after a remote delivery attempt has failed (for example, a connection timeout), each line in the retry configuration is first tested against the remote host name, and then against the domain name in the address. For example, if the MX records for **a.b.c.d** are

```
a.b.c.d  MX  5  x.y.z
         MX  6  p.q.r
         MX  7  m.n.o
```

and the retry rules are

```
p.q.r    *      F,24h,30m;
a.b.c.d  *      F,4d,45m;
```

then failures to deliver to host **p.q.r** use the first rule to determine retry times, but for all the other hosts for the domain **a.b.c.d**, the second rule is used, and that rule would also be used if routing to **a.b.c.d** suffers a temporary failure.

The second field in a retry rule is the name of a particular error, or an asterisk, which matches any error. The errors that can be tested for are:

refused_MX: connection refused from a host obtained from an MX record

refused_A: connection refused from a host not obtained from an MX record

refused: any connection refusal

timeout_connect: connection timed out

timeout_DNS: DNS lookup timed out

timeout: any timeout

quota: quota exceeded in local delivery

quota_<*time*>: quota exceeded in local delivery, and the mailbox has not been read for <*time*>.

The quota errors apply both to system-enforced quotas and to Exim's own quota mechanism in the **appendfile** transport.

The third field in a retry rule is a sequence of retry parameter sets, separated by semicolons. Each set consists of

<*letter*>`,`<*cutoff time*>`,`<*arguments*>

The letter identifies the algorithm for computing a new retry time; the cutoff time is the time beyond which this algorithm no longer applies, and the arguments vary the algorithm's action. The cutoff time is measured from the time that the first failure for the domain (combined with the local part if relevant) was detected, not from the time the message was received. The available algorithms are:

- **F**: retry at fixed intervals. There is a single time parameter specifying the interval.
- **G**: retry at geometrically increasing intervals. The first argument specifies a starting value for the interval, and the second a multiplier.

When computing the next retry time, the algorithm definitions are scanned in order until one whose cutoff time has not yet passed is reached. This is then used to compute a new retry time that is later than the current time. In the case of fixed interval retries, this simply means adding the interval to the current time. For geometrically increasing intervals, retry intervals are computed from the rule's parameters until one that is greater than the previous interval is found. The main configuration variable **retry_interval_max** limits the maximum interval between retries.

A single remote domain may have a number of hosts associated with it, and each host may have more than one IP address. Retry algorithms are selected on the basis of the domain name, but are applied to each IP address independently. If, for example, a host has two IP addresses and one is broken, Exim will generate retry times for it and will not try to use it until its next retry time comes. Thus the good IP address is likely to be tried first most of the time.

Retry times are hints rather than promises. Exim does not make any attempt to run deliveries exactly at the computed times. Instead, a queue-running process starts delivery processes for delayed messages periodically, and these attempt new deliveries only for those addresses that have passed their next retry time. Therefore, whatever you set in the retry rules, the minimum time between retries is the interval between queue-running processes. There is not much point in setting retry times of five minutes if your queue-runners happen only once an hour.

31.2 Retry rule examples

Here are some example retry rules suitable for use when **wonderland.fict.book** is a local domain:

```
alice@wonderland.fict.book quota_5d  F,7d,3h
wonderland.fict.book       quota_5d
wonderland.fict.book       *         F,1h,15m; G,2d,1h,2;
lookingglass.fict.book     *         F,24h,30m;
*                          refused_A F,2h,20m;
*                          *         F,2h,15m; G,16h,1h,1.5; F,5d,8h
```

The first rule sets up special handling for mail to **alice@wonderland.fict.book** when there is an over-quota error and the mailbox hasn't been read for at least 5 days. Retries continue every three hours for 7 days. The second rule handles over-quota errors for all other local parts at **wonderland.fict.book**; the absence of a local part has the same effect as supplying '*@'. As no retry algorithms are supplied, messages that fail are bounced immediately if the mailbox hasn't been read for at least 5 days.

The third rule handles all other errors at **wonderland.fict.book**; retries happen every 15 minutes for an hour, then with geometrically increasing intervals until two days have passed since a delivery first failed. The fourth rule controls retries for the domain **lookingglass.fict.book**, whether it is local or remote, and the remaining two rules handle all other domains, with special action for connection refusal from hosts that were not obtained from an MX record.

The final rule in a retry configuration should always have asterisks in the first two fields so as to provide a general catch-all for any addresses that do not have their own special handling. This example tries every 15 minutes for 2 hours, then with intervals starting at one hour and increasing by a factor of 1.5 up to 16 hours, then every 8 hours up to 5 days.

31.3 Long-term failures

Special processing happens when an address has been failing for so long that the cutoff time for the last algorithm has been reached. This is independent of how long any specific message has been failing; it is the length of continuous failure for the address that counts. When this is the case for a local delivery, or for all IP addresses associated with a remote delivery, a subsequent delivery failure causes Exim to give up on the address, and a delivery error message is generated. In order to cater for new messages that may use the failing address, a next retry time is still computed from the final algorithm, and is used as follows:

If the delivery is a local one, one delivery attempt is always made for any subsequent messages. If it fails, the address fails immediately. The post-cutoff retry time is not used.

If the delivery is remote, there are two possibilities, controlled by the **delay_after_cutoff** option of the **smtp** transport. The option is true by default and in that case:

> Until the post-cutoff retry time for one of the IP addresses is reached, any attempt to deliver to the failing address is bounced immediately. After that time, one new delivery attempt is made to those IP addresses that are past their retry times, and if that still fails, the address is bounced and new retry times are computed.

In other words, Exim delays retrying an IP address after the final cutoff time until a new retry time is reached, and can therefore bounce an email address without ever trying a delivery when machines have been down for a long time. This ensures that few resources are wasted in repeatedly trying to deliver to a broken destination, but if it does recover, Exim will eventually notice.

If **delay_after_cutoff** is set false, Exim behaves differently. If all IP addresses are past their final cutoff time, Exim tries to deliver to those IP addresses that have not been tried since the message arrived. If there are none, or if they all fail, the address is bounced. In other words, it does not delay when a new message arrives, but tries the expired addresses immediately, unless they have been tried since the message arrived. If there is a continuous stream of messages for the failing domains, unsetting **delay_after_cutoff** means that there will be many more attempts to deliver to failing IP addresses than when **delay_after_cutoff** is true.

An additional rule is needed to cope with cases where a host is intermittently available, or when a message has some attribute that prevents its delivery when others to the same address get through. Because some messages are successfully delivered, the ‘retry clock’ keeps getting restarted, and so a message could remain on the queue for ever. To prevent this, if a message has been on the queue for longer than the cutoff time of any applicable retry rule, the associated email address is failed after its next temporary delivery error. A new retry time is not computed in this case, so that other messages for the same address are considered immediately.

Even with this rule a large queue of messages can take a long time to clear if some occasionally get delivered, because the intermittent failures delay delivery attempts on the others (and the above rule acts only after a delivery attempt). There is therefore an ultimate clean-up rule which causes all the remaining addresses in a message to be failed, whether or not there has just been a delivery attempt, if the message has been on the queue for longer than the longest cutoff time for any retry rule in the configuration file.

The data in the retry hints database can be inspected by using the **exim_dumpdb** or **exim_fixdb** utility programs (see chapter 47). The latter utility can also be used to change the data. The **exinext** utility script can be used to find out what the next retry times are for the hosts associated with a particular mail domain, and also for local deliveries that have been deferred.

32. Address rewriting

Some people believe that configured address rewriting is a Mortal Sin. Others believe that life is not possible without it. Exim provides the facility; you do not have to use it. There are two cases that are commonly encountered:

- The company **hitch.fict.book** has a number of machines that exchange mail with each other behind a firewall, but only a single gateway to the outer world. The gateway rewrites **∗.hitch.fict.book** as **hitch.fict.book**.
- A machine rewrites the local parts of its own users so that, for example, **fp42@hitch.fict.book** becomes **Ford.Prefect@hitch.fict.book**.

In general, rewriting addresses from one's own system or domain has some legitimacy. Rewriting other addresses should be done only with great care and in special circumstances.

Address rewriting can be applied both to envelope addresses and addresses in header lines. Exim's rewriting rules specify to which addresses they apply.

- Rewriting rules for envelope addresses are also applied to new addresses that are generated by aliasing or forwarding operations, unless **no_rewrite** is set on the relevant directors.
- Rewriting of headers happens when a message is received, which is why it is possible to rewrite **Bcc:** headers (Exim removes **Bcc:** headers only when the **-t** option is used). It does not apply to headers that are added by the generic driver option **add_headers**.

Exim's rewriting configuration appears as the sixth part of the runtime configuration file. It can be tested by the **-brw** command line option. This takes an address (which can be a full RFC 822 address) as its argument. The output is a list of how the address would be transformed by the rewriting rules for each of the different places it might appear, that is, for each different header and for the envelope sender and recipient fields. For example,

```
exim -brw ph10@exim.work.shop
```

might produce the output

```
  sender: Philip.Hazel@exim.work.shop
    from: Philip.Hazel@exim.work.shop
      to: ph10@exim.work.shop
      cc: ph10@exim.work.shop
     bcc: ph10@exim.work.shop
reply-to: Philip.Hazel@exim.work.shop
env-from: Philip.Hazel@exim.work.shop
  env-to: ph10@exim.work.shop
```

which shows that rewriting has been set up for that address when used in any of the source fields, but not when it appears as a recipient address.

32.1 Rewriting rules

The rewriting configuration consists of lines of rewriting rules in the form

<*source pattern*>   <*replacement*>   <*flags*>

The flags are single characters which may appear in any order. Spaces and tabs between them are ignored.

The formats of source patterns and replacement strings are described below. Each is terminated by white space. If a replacement string contains spaces, which can happen for certain forms of expansion expression, it must be enclosed in double quotes, and the normal quoting conventions apply inside them.

Long rules (whether using quotes or not) can be split over several lines by terminating all but the last with a backslash character. Leading white space on continuation lines is ignored. Consequently, a continuation backslash should never appear immediately after the source pattern or replacement string, but instead should follow its terminating white space.

For each address that could potentially be rewritten, the rules are scanned in order, and replacements from earlier rules can themselves be replaced as a result of later rules (but see the ‘q’ and ‘R’ flags).

The order in which header and envelope addresses are rewritten is undefined, may change between releases, and must not be relied on. For example, the replacement string for a header rewrite must not assume that the message’s envelope sender address has (or has not) already been rewritten.

$local_part and **$domain** can be used in the replacement string to refer the address that is being rewritten. Note that complete lookup-driven rewriting can be done by a line of the form

```
*@*   ${lookup ...
```

where the lookup key is derived from **$1** and **$2** or **$local_part** and **$domain**.

32.2 Rewriting patterns

The source pattern can be in one of the following forms. Note that it is not enclosed in quotes, and there is no special processing of any characters. In particular, if it is a regular expression, backslash characters should not be doubled.

- An address containing a local part and a domain, either of which may start with an asterisk, implying independent wildcard matching, for example

  ```
  *@orchestra-land.fict.book
  ```

  If the domain is specified as a single @ character, it matches the primary host name. After matching, the numerical variables refer to the character strings matched by asterisks, with **$1** associated with the first asterisk, while **$0** refers to the entire address. For example, if the pattern

  ```
  *queen@*.fict.book
  ```

  is matched against the address **hearts-queen@wonderland.fict.book** then

  $0 = **hearts-queen@wonderland.fict.book**
  $1 = **hearts-**
  $2 = **wonderland**

  Note that if the local part does not start with an asterisk, but the domain does, then it is **$1** that contains the wild part of the domain.

- A local part, possibly starting with an asterisk, and a lookup item (as in a domain list), for example

  ```
  root@lsearch;/special/domains
  ```

  If there is an asterisk in the local part, the value of the wild part is placed in the first numerical variable. If the lookup is a partial one, the wild part of the domain is placed in the next numerical variable, and the fixed part of the domain is placed in the succeeding variable. Thus, for example, if the address **foo@bar.baz.com** is processed by a rewriting rule of the form

  `*@partial-dbm;/some/dbm/file`   <*replacement string*>

  and the key in the file that matches the domain is ***.baz.com**, then

  $1 = **foo**
  $2 = **bar**
  $3 = **baz.com**

  If the address **foo@baz.com** is looked up, this matches the same wildcard file entry, and in this case **$2** is set to the empty string, but **$3** is still set to **baz.com**. If a non-wild key is matched in a partial lookup, then again **$2** is set to the empty string and **$3** is set to the whole domain. For non-partial lookups, no numerical variables are set.

- A local part, possibly starting with an asterisk, and a regular expression (as in a domain list), for example

  ```
  *.queen@^(wonderland|lookingglass).fict.book$
  ```

  If there is an asterisk in the local part, the value of the wild part is placed in the first numerical variable. Any substrings captured by the regular expression are placed in numerical variables starting at **$1** if there is no asterisk in the local part, or at **$2** if there is.

- A lookup without a local part, for example

  ```
  partial-dbm;/rewrite/database
  ```

  This works as for an *address-list* configuration item – the domain is first looked up, possibly partially, and if that fails, the whole address is then looked up (not partially). When a partial lookup succeeds, the numerical variable **$1** contains the wild part of the domain, and **$2** contains the fixed part. The '@@' form of address-list lookup can also be used.

- A single regular expression. This is matched against the entire address, with the domain part lower-cased. After matching, the numerical variables refer to the bracketed 'capturing' sub-expressions, with **$0** referring to the entire address. For example, if the pattern

  ```
  ^(red|white).king@(wonderland|lookingglass).fict.book$
  ```

  is matched against the address **red.king@lookingglass.fict.book** then

  `$0 =` **red.king@lookingglass.fict.book**
  `$1 =` **red**
  `$2 =` **lookingglass**

  Note that because the pattern part of a rewriting rule is terminated by white space, no white space may be present in the regular expression.

32.3 Rewriting replacements

If the replacement string for a rule is a single asterisk, then addresses that match the pattern and flags are *not* rewritten, and no subsequent rewriting rules are scanned. For example,

```
hatta@lookingglass.fict.book  *  f
```

specifies that **hatta@lookingglass.fict.book** is never to be rewritten in **From:** headers.

Otherwise, the replacement string is expanded and must yield a fully qualified address. Within the expansion, the variables **$local_part** and **$domain** refer to the address that is being rewritten. Any letters they contain retain their original case – they are not lower cased. The numerical variables are set up according to the type of pattern that matched the address, as described above. If the expansion is forced to fail by the presence of 'fail' in a conditional or lookup item, rewriting by the current rule is abandoned. Any other expansion failure causes the entire rewriting operation to be abandoned, and an entry written to the panic log.

32.4 Rewriting flags

There are four different kinds of flag that may appear on rewriting rules:

- Flags that specify which headers and envelope addresses to rewrite: E, F, T, b, c, f, h, r, s, t.
- A flag that specifies rewriting at SMTP time: S.
- Flags that control the rewriting process: Q, q, R, w.
- A special-purpose flag for additional relay checking: X.

32.5 Flags specifying which headers and envelope addresses to rewrite

If none of the following flag letters, nor the 'S' flag (see section 32.6) are present, the rewriting rule applies to all headers and to both the sender and recipient fields of the envelope. Otherwise, the rewriting rule is skipped unless the relevant addresses are being processed.

`E`	rewrite all envelope fields
`F`	rewrite the envelope From field
`T`	rewrite the envelope To field
`b`	rewrite the **Bcc:** header
`c`	rewrite the **Cc:** header
`f`	rewrite the **From:** header
`h`	rewrite all headers
`r`	rewrite the **Reply-to:** header
`s`	rewrite the **Sender:** header
`t`	rewrite the **To:** header

You should be particularly careful about rewriting **Sender:** headers, and restrict this to special known cases in your own domains.

32.6 The SMTP-time rewriting flag

The rewrite flag 'S' specifies a rewrite at SMTP time, as soon as an address is received and before any other processing; even before syntax checking. The pattern is required to be a regular expression. This applies to both sender and recipient addresses, and allows for the handling of addresses that are not compliant with RFC 822 (for example, 'bang paths' in batched SMTP input). Because of this, the variables **$local_part** and **$domain** are not available during the expansion of the replacement string.

32.7 Flags controlling the rewriting process

There are four flags which control the way the rewriting process works. These take effect only when a rule is invoked, that is, when the address is of the correct type (matches the flags) and matches the pattern.

- If the 'Q' flag is set on a rule, the rewritten address is permitted to be an unqualified local part. It is qualified with **qualify_recipient**. In the absence of 'Q' the rewritten address must always include a domain.
- If the 'q' flag is set on a rule, then no further rewriting rules are considered, even if no rewriting actually takes place because of a 'fail' in the expansion. The 'q' flag is not effective if the address is of the wrong type (does not match the flags) or does not match the pattern.
- The 'R' flag causes a successful rewriting rule to be re-applied to the new address, up to ten times. It can be combined with the 'q' flag, to stop rewriting once it fails to match (after at least one successful rewrite).
- When an address in a header is rewritten, the rewriting normally applies only to the working part of the address, with any comments and RFC 822 'phrase' left unchanged. For example, rewriting might change

  ```
  From: Ford Prefect <fp42@restaurant.hitch.fict.book>
  ```

  into

  ```
  From: Ford Prefect <prefectf@hitch.fict.book>
  ```

  Sometimes there is a need to replace the whole address item, and this can be done by adding the flag letter 'w' to a rule. If this is set on a rule that causes an address in a header to be rewritten, the entire address is replaced, not just the working part. The replacement must be a complete RFC 822 address, including the angle brackets if necessary. When the 'w' flag is set on a rule that causes an envelope address to be rewritten, all but the working part of the replacement address is discarded.

32.8 The additional relay checking flag

The ‘X’ flag is a slightly strange oddity that adds additional checking to **sender_address_relay**. Whenever an address passes the **sender_address_relay** check, if there are any rewriting rules with the ‘X’ flag set, the address is rewritten and if this makes any change to the address, it must verify successfully for the relaying to be permitted.

We use this in Cambridge as follows: users have a centrally registered address in the virtual domain **cam.ac.uk**, but there are a number of different hosts where they actually have their accounts and from which they can read mail using IMAP or POP. It is desirable to prevent them using hosts other than those on which they have accounts as outgoing relays, and yet to permit the sending addresses to contain the **cam.ac.uk** domain. Since the user names are the same on the relay hosts as in the **cam.ac.uk** domain, a rewriting rule of the form

```
*@cam.ac.uk  $1@${qualify_domain}  X
```

means that any sender address of the form **user@cam.ac.uk** is acceptable only if **user** has an account on the local host. This also has the virtue of detecting typos in the configurations of users’ MUAs.

32.9 Rewriting examples

Here is an example of the two common rewriting paradigms:

```
*@*.hitch.book.fict  $1@hitch.book.fict
*@hitch.book.fict    ${lookup{$1}dbm{/etc/realnames}\
                     {$value}fail}@hitch.book.fict bctfrF
```

Note the use of ‘fail’ in the lookup expansion. This causes the string expansion to fail, and in this context it has the effect of leaving the original address unchanged, but Exim goes on to consider subsequent rewriting rules, if any, since the ‘q’ flag is not present in that rule. An alternative to ‘fail’ would be to supply **$1** explicitly, which would cause the rewritten address to be the same as before, at the cost of a small bit of processing. Not supplying either of these is an error, since the rewritten address would then contain no local part.

The first example above replaces the domain with a superior, more general domain. This may not be desirable for certain local parts. If the rule

```
root@*.hitch.book.fict  *
```

were inserted as the first rule, rewriting is suppressed if the local part is **root** at any domain ending in **hitch.book.fict**.

Rewriting can be made conditional on a number of tests, by making use of **${if** in the expansion item. For example, to apply a rewriting rule only to messages that originate outside the local host:

```
*@*.hitch.book.fict  "${if !eq {$sender_host_address}{}\
                      {$1@hitch.book.fict}fail}"
```

The replacement string is quoted in this example because it contains white space.

Exim does not handle addresses in the form of ‘bang paths’. If it sees such an address it treats it as an unqualified local part which it qualifies with the local qualification domain (if the source of the message is local or if the remote host is permitted to send unqualified addresses). Rewriting can sometimes be used to handle simple bang paths with a fixed number of components. For example, the rule

```
^([^!]+)!(.*)@your\.domain$   $2@$1
```

rewrites a two-component bang path ‘host.name!user’ as the domain address ‘user@host.name’. However, there is a security implication in doing this as a normal rewriting rule for envelope addresses. It can provide a backdoor method for using your system as a relay, since the incoming addresses appear to be local. If the bang path addresses are received via SMTP, it is safer to use the ‘S’ flag to rewrite them as they are received, so that relay checking can be done on the rewritten addresses.

33. Customizing error and warning messages

When a message fails to get delivered, or remains on the queue for more than a configured amount of time, Exim sends a message to the original sender, or to an alternative configured address. The text of these messages is built into the code of Exim, but it is possible to change it, either by adding a single string, or by replacing each of the paragraphs by text supplied in a file.

33.1 Customizing error messages

If **errmsg_text** is set, its contents are included in the default message immediately after 'This message was created automatically by mail delivery software.' The string is not expanded. It is not used if **errmsg_file** is set.

When **errmsg_file** is set, it must point to a template file for constructing error messages. The file consists of a series of text items, separated by lines consisting of exactly four asterisks. If the file cannot be opened, default text is used and a message is written to the main and panic logs. If any text item in the file is empty, default text is used for that item.

Each item of text that is read from the file is expanded, and there are two expansion variables which can be of use here: **$errmsg_recipient** is set to the recipient of an error message while it is being created, and **$return_size_limit** contains the value of the **return_size_limit** option, rounded to a whole number.

The items must appear in the file in the following order:

- The first item is included in the headers, and should include at least a **Subject:** header. Exim does not check the syntax of these headers.
- The second item forms the start of the error message. After it, Exim lists the failing addresses with their error messages.
- The third item is used to introduce any text from pipe transports that is to be returned to the sender. It is omitted if there is no such text.
- The fourth item is used to introduce the copy of the message that is returned as part of the error report.
- The fifth item is added after the fourth one if the returned message is truncated because it is bigger than **return_size_limit**.
- The sixth item is added after the copy of the original message.

The default state (**errmsg_file** unset) is equivalent to the following file, in which the sixth item is empty. The **Subject:** line has been split into two here in order to fit it on the page.

```
Subject: Mail delivery failed
  ${if eq{$sender_address}{$errmsg_recipient}{: returning message to sender}}
****
This message was created automatically by mail delivery software.

A message ${if eq{$sender_address}{$errmsg_recipient}{that you sent }{sent by

  <$sender_address>

}}could not be delivered to all of its recipients.
The following address(es) failed:
****
The following text was generated during the delivery attempt(s):
****
------ This is a copy of the message, including all the headers. ------
****
------ The body of the message is $message_size characters long; only the first
```

```
------ $return_size_limit or so are included here.
****
```

33.2 Customizing warning messages

The option **warnmsg_file** can be pointed at a template file for use when warnings about message delays are created. In this case there are only three text sections:

- The first item is included in the headers, and should include at least a **Subject:** header. Exim does not check the syntax of these headers.
- The second item forms the start of the warning message. After it, Exim lists the delayed addresses.
- The third item then ends the message.

The default state is equivalent to the file

```
Subject: Warning: message $message_id delayed $warnmsg_delay
****
This message was created automatically by mail delivery software.

A message ${if eq{$sender_address}{$warnmsg_recipients}{that you sent }{sent by

  <$sender_address>

}}has not been delivered to all of its recipients after
more than $warnmsg_delay on the queue on $primary_hostname.

The message identifier is:     $message_id
The subject of the message is: $h_subject
The date of the message is:    $h_date

The following address(es) have not yet been delivered:
****
No action is required on your part. Delivery attempts will continue for
some time, and this warning may be repeated at intervals if the message
remains undelivered. Eventually the mail delivery software will give up,
and when that happens, the message will be returned to you.
```

except that in the default state the subject and date lines are omitted if no appropriate headers exist. During the expansion of this file, **$warnmsg_delay** is set to the delay time in one of the forms '<*n*> minutes' or '<*n*> hours', and **$warnmsg_recipients** contains a list of recipients for the warning message. There may be more than one if there are multiple addresses with different **errors_to** settings on the routers/directors that handled them.

34. The default configuration file

The default configuration file supplied with Exim as **src/configure.default** is sufficient for a single host with simple mail requirements. It contains comments about options you might want to set, but which it lets default, together with the settings described here.

34.1 Main configuration settings

There are two explicit options in this section:

```
never_users = root
```

This prevents Exim from ever running as root when performing a local delivery. Instead, it runs as ‘nobody’.

```
host_lookup = 0.0.0.0/0
```

This specifies the sending IP networks for which a DNS reverse lookup is done, in order to get the host name from the IP address of an incoming message. The default setting matches all IP addresses. The host name appears in the log and in messages’ **Received:** headers.

As the **primary_hostname**, **qualify_domain**, and **local_domains** options are not specified, they all take the name of the local host, as obtained by the **uname()** function, as their value.

No relaying is permitted through the host, because neither **relay_domains** nor **host_accept_relay** is set. See chapter 40 for more details about relay control.

34.2 Transport configuration settings

Four local transports and one remote transport are defined. The first o ne is the remote transport:

```
remote_smtp:
  driver = smtp
```

This transport is used to do external deliveries over SMTP, with default options. The first local transport is

```
local_delivery:
  driver = appendfile
  file = /var/mail/${local_part}
  delivery_date_add
  envelope_to_add
  return_path_add
```

This is set up to deliver to local mailboxes in a traditional ‘sticky bit’ directory. Some installations prefer not to set the ‘sticky bit’, but instead run the delivery under a specific group, with the directory being writeable by the group. Adding the following options achieves this:

```
   group = mail
   mode = 0660
```

To deliver into files in users’ home directories, a setting such as

```
  file = /home/${local_part}/inbox
```

or

```
  file = ${home}/inbox
```

should be substituted for the default **file** option. The three options ending in **_add** cause Exim to add three header lines to the message as it writes it to the mailbox. They can be removed if these headers are not required. The second local transport is

```
address_pipe:
  driver = pipe
  return_output
```

This transport is used by Exim when a local part that is expanded via an alias or forward file causes delivery to a pipe. Any output from the pipe is returned to the sender of the message. The third local transport is

```
address_file:
  driver = appendfile
  delivery_date_add
  envelope_to_add
  return_path_add
```

This transport is used by Exim when a local part that is expanded via an alias or forward file causes delivery to a specified file (by generating a path name not ending in '/'). The final local transport is

```
address_reply:
  driver = autoreply
```

This transport used by Exim when a local part that is expanded via a filter file causes an automatic reply to a message to be generated.

34.3 Director configuration settings

Three directors are specified for the default configuration. Note that the order of director definitions matters. The first director causes local parts to be checked against the system alias file, which is searched linearly:

```
system_aliases:
  driver = aliasfile
  file = /etc/aliases
  search_type = lsearch
  file_transport = address_file
  pipe_transport = address_pipe
```

If an alias generates a file or pipe delivery, the **address_file** or **address_pipe** transport is used, as appropriate. The second director comes into play if a local part does not match a system alias:

```
userforward:
  driver = forwardfile
  file = .forward
  no_verify
  no_expn
  check_ancestor
# filter
  filetransport = addressfile
  pipetransport = addresspipe
  replytransport = addressreply
```

An attempt is made to look for a file called **.forward** in the home directory of a local user. However, this director is skipped when verifying addresses or running an SMTP EXPN command. The **check_ancestor** option prevents a **.forward** file from turning a login name back into a previously-handled alias name. The **filter** option is commented out in the default configuration. Thus **.forward** files are treated in the conventional manner, but filtering can be enabled by removing the # character.

If forwarding or filtering generates a file, pipe, or autoreply delivery, the **address_file**, **address_pipe**, or **address_reply** transport is used, as appropriate. The final director is

```
localuser:
  driver = localuser
  transport = local_delivery
```

This checks that a local part is the login of a local user, and if so, directs the message to be delivered using the **local_delivery** transport.

34.4 Router configuration settings

Two routers are defined in the default configuration. The first is

```
lookuphost:
  driver = lookuphost
  transport = remote_smtp
```

and its default settings cause it to look up the domain in the DNS, in order to determine the host to which a message should be sent, using the **remote_smtp** transport. The second router is

```
literal:
  driver = ipliteral
  transport = remote_smtp
```

This handles ‘domains’ that are actually RFC 822 domain literals, that is, IP addresses enclosed in square brackets.

34.5 Default retry rule

A single retry rule is given in the default configuration:

```
*     *    F,2h,15m; G,16h,1h,1.5; F,4d,8h
```

This causes any temporarily failing address to be retried every 15 minutes for 2 hours, then at intervals starting at one hour and increasing by a factor of 1.5 until 16 hours have passed, then every 8 hours up to 4 days.

34.6 Rewriting configuration

There are no rewriting rules in the default configuration file.

35. Multiple user mailboxes

The wildcard facility of the generic **prefix** and **suffix** options for directors allows you to configure Exim to permit users to make use of arbitrary local part prefixes or suffixes in any way they wish. A director such as

```
userforward:
  driver = forwardfile
  file = .forward
  suffix = -*
  suffix_optional
  filter
```

runs a user's **.forward** file for all local parts of the form **username-***. Within the filter file the user can distinguish different cases by testing the variable **$local_part_suffix**. For example:

```
if $local_part_suffix contains -special then
  save /home/$local_part/Mail/special
endif
```

If the filter file does not exist, or does not deal with such addresses, they fall through to subsequent directors, and, assuming no subsequent use of the **suffix** option is made, they presumably fail. Thus users have control over what suffixes are valid.

Alternatively, a suffix can be used to trigger the use of a different **.forward** file – which is the way a similar facility is implemented in another MTA:

```
userforward:
  driver = forwardfile
  file = .forward${local_part_suffix}
  suffix = -*
  suffix_optional
  filter
```

If there is no suffix, **.forward** is used; if the suffix is **-special**, for example, then **.forward-special** is used. Once again, if the appropriate file does not exist, or does not deal with the address, it is passed on to subsequent directors, which could, if required, look for an unqualified **.forward** file to use as a default.

36. Using Exim to handle mailing lists

Exim can be used to run simple mailing lists, but for large and/or complicated requirements, the use of additional specialized mailing list software is recommended.

The **forwardfile** director can be used to handle mailing lists where each list is maintained in a separate file, which can therefore be managed by an independent manager. The **domains** director option can be used to run these lists in a separate domain from normal mail. For example:

```
lists:
  driver = forwardfile
  domains = lists.ref.book
  no_more
  file = /opt/lists/${local_part}
  no_check_local_user
  forbid_pipe
  forbid_file
  errors_to = ${local_part}-request@lists.ref.book
```

The domain **lists.ref.book** must appear as one of the domains in the **local_domains** configuration option. This director is used only when an address refers to that domain. Because the **no_more** option is set, if the local part of the address does not match a file in the **/opt/lists** directory, causing the director to fail, no subsequent directors are tried, and the whole delivery fails.

The **no_check_local_user** option stops Exim insisting that the local part is the login id of a local user, and because no user or group is specified, no check is made on the ownership of the file. The **forbid_pipe** and **forbid_file** options prevent a local part from being expanded into a file name or a pipe delivery.

The **errors_to** option specifies that any delivery errors caused by addresses taken from a mailing list are to be sent to the given address rather than the original sender of the message. However, before acting on this, Exim verifies the error address, and ignores it if verification fails.

For example, using the configuration above, mail sent to **dicts@lists.ref.book** is passed on to those addresses contained in **/opt/lists/dicts**, with error reports directed to **dicts-request@lists.ref.book**, provided that this address can be verified. There could be a file called **/opt/lists/dicts-request** containing the address(es) of this particular list's manager(s), but other approaches, such as setting up an earlier director (possibly using the **prefix** or **suffix** options) to handle addresses of the form **owner-xxx** or **xxx-request**, are also possible.

36.1 Syntax errors in mailing lists

If an entry in a forward file contains a syntax error, Exim normally defers delivery of the entire message. This may not be appropriate when the list is being maintained automatically from address texts supplied by users. If the **skip_syntax_errors** option is set on the **forwardfile** director, it just skips entries that fail to parse, noting the incident in the log. If in addition **syntax_errors_to** is set to a verifyable address, messages about skipped addresses are sent to it.

36.2 NFS-mounted mailing lists

It is not advisable to have list files that are NFS mounted, since the absence of the mount cannot be distinguished from a non-existent file. One way round this is to use an **aliasfile** director where the alias file is local and contains a list of the lists, and each alias expansion is simply an 'include' item to get the list from a separate, NFS mounted file. If **no_freeze_missing_include** is set for the **aliasfile** director, an unavailable file then just causes delivery to be deferred.

36.3 Re-expansion of mailing lists

Exim remembers every individual address to which a message has been delivered, in order to avoid duplication, but it normally stores only the original recipient addresses with a message. If all the deliveries to a mailing list cannot be done at the first attempt, the mailing list is re-expanded when the delivery is next tried. This means that alterations to the list are taken into account at each delivery attempt, and addresses that have been added to the list since the message arrived will thus receive a copy of the message, even though it pre-dates their subscription.

If this behaviour is felt to be undesirable, the **one_time** option can be set on the **forwardfile** director. If this is done, any addresses generated by the director that fail to deliver at the first attempt are added to the message as 'top level' addresses, and the parent address that generated them is marked 'delivered'. Thus expansion of the mailing list does not happen again at the subsequent delivery attempts. The disadvantage of this is that if any of the failing addresses are incorrect, correcting them in the file has no effect on pre-existing messages.

The original top-level address is remembered with each of the generated addresses, and is output in any log messages. However, any intermediate parent addresses are not recorded. This makes a difference to the log only if **log_all_parents** is set, but for mailing lists there is normally only one level of expansion anyway.

36.4 Closed mailing lists

The examples so far have assumed open mailing lists, to which anybody may send mail. It is also possible to set up closed lists, where mail is accepted from specified senders only. This is done by making use of the generic **senders** option. The following example uses the same file for each list, both as a list of recipients and as a list of permitted senders. In this case, it is necessary to set up a separate director to handle the '-request' address.

```
# Handle mail to xxx-request@lists.ref.book;
# anybody can mail to this address.

lists_request:
  driver = forwardfile
  domains = lists.ref.book
  suffix = -request
  file = /opt/lists/${local_part}${local_part_suffix}
  no_check_local_user

# Handle mail to xxx@lists.ref.book;
# only the subscribers to a list may mail to it.
# Use one_time to prevent multiple expansions.

lists:
  driver = forwardfile
  domains = lists.ref.book
  no_more
  require_files = /opt/lists/${local_part}
  senders = lsearch;opt/lists/${local_part}
  file = /opt/lists/${local_part}
  no_check_local_user
  forbid_pipe
  forbid_file
  one_time
  skip_syntax_errors
  errors_to = ${local_part}-request@lists.ref.book
```

The **require_files** option is needed to ensure that the file exists before trying to search it via the **senders** option; an attempt to search a non-existent file causes Exim to panic. If the file does not exist – that is, if the mailing list is unknown, the director fails, but because **no_more** is set, no further directors are tried, and so Exim gives up.

37. Virtual domains

There are a number of ways in which virtual domains can be handled in Exim. As this seems to be quite a common requirement, some ways of doing this are described here. These are not the only possibilities.

37.1 All mail to a given host

Simply sending all mail for a domain to a given host isn't really a virtual domain; it is just a routing operation that can be handled by a **domainlist** router.

To send all mail for a domain to a particular local part at a given host, define the domain as local, then process it with a **smartuser** director that sets the new delivery address and passes the message to an **smtp** transport which specifies the host. Alternatively, use a **forwardfile** director pointing to a fixed file name; the file can contain any number of addresses to which each message is forwarded.

37.2 Virtual domains not preserving envelopes

A virtual domain that does not preserve the envelope information when delivering can be handled by an alias file defined for a local domain. If you are handling a large number of local domains, you can define them as a file lookup. For example:

```
local_domains = "your.normal.domain:\
                 dbm;/customer/domains"
```

Where **/customer/domains** is a DBM file built from a source file that contains just a list of domains:

```
# list of virtual domains for customers
customer1.domain
customer2.domain
```

This can be turned into a DBM file by **exim_dbmbuild**.

You can then set up a director (see below) to handle the customer domains, arranging a separate alias file for each domain. A single director can handle all of them if the names follow a fixed pattern. Permissions can be arranged so that appropriate people can edit the alias files. The **domains** option ensures that this director is used only for the customer domains. The DBM file lookup is cached, so it isn't too inefficient to do this. The **no_more** setting ensures that if the lookup fails, Exim gives up on the address without trying any subsequent directors.

```
virtual:
  driver = aliasfile
  domains = dbm;/customer/domains
  no_more
  file = /etc/mail/$domain
  search_type = lsearch
```

A successful aliasing operation results in a new envelope recipient address, which is then directed or routed from scratch.

37.3 Virtual domains preserving envelopes

If you want to arrange for mail for known local parts at certain domains to be sent to specific hosts without changing the envelope recipients of messages, then the following is one way of doing it.

Set up the domains as local, and create an **aliasfile** director for them, as above, but in addition, specify a transport for the director:

```
  virtual:
    driver = aliasfile
    domains = dbm;/customer/domains
    transport = virtual_smtp
    no_more
    file = /etc/mail/$domain
    search_type = lsearch
```

Each domain has its own alias file, but the provision of a transport means that this is used purely as a check list of local parts. The data portion of each alias is not used.

The transport has to look up the appropriate host to which the message must be sent:

```
  virtual_smtp:
    driver = smtp
    hosts = ${lookup{$domain}dbm{/virtual/routes}{$value}fail}
```

The file **/virtual/routes** contains lines of the form

```
  customer1.domain:  cust1.host
  customer2.domain:  cust2.host
```

and the messages get delivered with RCPT (the envelope) containing the original destination address (for example, **postmaster@customer1.domain**). In fact, you could use the same file for **/virtual/routes** and **/customer/domains**, since the lookup on the latter doesn't make any use of the data – it's just checking that the file contains the key.

38. Intermittently connected hosts

It is becoming quite common (because it is cheaper) for hosts to connect to the Internet periodically rather than remain connected all the time. The normal arrangement is that mail for such hosts accumulates on a system that is permanently connected.

Exim was designed for use on permanently connected hosts, and so it is not particularly well-suited to use in an intermittently connected environment. Nevertheless there are some features that can be used.

38.1 Exim on the upstream host

If the ‘holding system’ is running Exim, then it should be configured with a long retry period for the intermittent host. For example:

```
  cheshire.wonderland.fict.book    *   F,5d,24h
```

This stops a lot of failed delivery attempts from occurring, but Exim remembers which messages it has queued up for that host. Once the intermittent host comes online, forcing delivery of one message (either by using the **-M** or **-R** options, or by using the ETRN SMTP command – see **smtp_etrn_hosts** and section 42.6) causes all the queued up messages to be delivered, often down a single SMTP connection. While the host remains connected, any new messages get delivered immediately.

If the connecting hosts do not have fixed IP addresses, that is, if a host is issued with a different IP address each time it connects, then Exim’s retry mechanisms on the holding host get confused, because the IP address is normally used as part of the key string for holding retry information. This can be avoided by unsetting **retry_include_ip_address** on the **smtp** transport. Since this has disadvantages for permanently connected hosts, it is best to arrange a separate transport for the intermittently connected ones.

38.2 Exim on the intermittently connected host

The value of **smtp_accept_queue_per_connection** should probably be increased, or even set to zero (that is, disabled) on the intermittently connected host, so that all incoming messages down a single connection get delivered immediately.

Mail waiting to be sent from an intermittently connected host will probably not have been routed, since without a connection DNS lookups are not possible. This means that if a normal queue run is done at connection time, each message is likely to be sent in a separate SMTP session. This can be avoided by starting the queue run with a command line option beginning with **-qq** instead of **-q**. In this case, the queue is scanned twice. In the first pass, routing is done but no deliveries take place. The second pass is a normal queue run; since all the messages have been previously routed, those destined for the same host are likely to get sent as multiple deliveries in a single SMTP connection.

38.3 Handling many intermittently connected hosts

Leaving mail for intermittently connected hosts on the main queue of a holding system as suggested above does not scale very well. Two different kinds of waiting message are being mixed up in the same queue – those that cannot be delivered because of some temporary problem, and those that are waiting for their destination host to connect. This makes it hard to manage the queue, as well as wasting resources, because each queue runner scans the entire queue.

A better approach is to separate off those messages that are waiting for an intermittently connected host. This can be done by using a separate version of Exim that stores only those messages, or by delivering such messages into local files in ‘mailstore’ or other envelope-preserving format, from where they are transmitted by other software when their destination connects. This makes it easy to collect all the mail for one host in a single directory, and to apply local timeout rules on a per-message basis if required.

39. Verification of incoming mail

Exim always checks the syntax of SMTP commands, and rejects any that are invalid. There are a number of options that cause Exim to verify the semantic validity of the data in an incoming SMTP message. Verification failures can cause the message to be rejected, or they can just be logged. Other types of control over incoming mail are discussed in subsequent chapters. The **-bh** command line option can be used to run fake SMTP sessions for the purpose of testing verification options.

39.1 Host verification

The name of the sending host is looked up using **gethostbyaddr()** if its IP address matches **host_lookup** (which is unset in the Exim binary, but in the default configuration file is set to match all hosts). In some environments this might involve an expensive DNS lookup, so some sites may wish to disable it. However, an SMTP server for local desktop systems (which are frequently misconfigured) can normally look up their host names cheaply. This improves the contents of Exim's logs by including the correct host names.

Even if its address doesn't match **host_lookup**, a sending host's real name is looked up from its IP address if the argument it provides for the HELO or EHLO command is the local host's own name, or the name of one of its local domains, which seems to be a fairly common misconfiguration.

A host name that is obtained from looking up the sender's IP address is placed in the **$sender_host_name** variable. If no lookup was done, or if the lookup failed, that variable is left empty. Failure to look up the sending host's name is not of itself an error, nor is it by default an error for the name given in the HELO or EHLO command (which is placed in **$sender_helo_name**) to be different.

The RFCs specifically state that mail should not be refused on the basis of the content of the HELO or EHLO commands. However, there are installations that do want to be strict in this area, and to support them, Exim has the **helo_verify** option. Even when this is not set, Exim checks the syntax of the commands, and rejects them if there are syntax errors. It can be made less strict by unsetting **helo_strict_syntax** (which allows underscores to get through) or by setting **helo_accept_junk_hosts** (which permits certain hosts to send any old junk).

When **helo_verify** is set, a HELO or EHLO command must precede any MAIL commands in an incoming SMTP connection. If there wasn't one, all MAIL commands are rejected with a permanent error code. In addition, the argument supplied by HELO or EHLO is verified. If it is in the form of a literal IP address in square brackets, it must match the actual IP address of the sending host. If it is a domain name, then the sending host's name is looked up from its IP address (whether or not it matches **host_lookup**) and compared against it. If the comparison fails, the IP addresses associated with the HELO or EHLO name are looked up using **gethostbyname()** and compared against the sending host's IP address. If none of them match, the HELO or EHLO command is rejected with a permanent error code, and an entry is written in the main and reject logs.

39.2 Sender verification

When **sender_verify** is set, Exim checks the senders of incoming SMTP messages, that is, the addresses given in the SMTP MAIL commands. This does not apply to batch SMTP input by default, but **sender_verify_batch** can be set true if it is required.

The check is performed by running the same verification code as is used then Exim is called with the **-bv** option. The check is performed when the MAIL command is received. If the address cannot immediately be verified (typically because of DNS timeouts), a temporary failure error response (code 451) is given after the data for the message has been received. It is delayed until this time so that the message's headers can be logged. However, if **sender_try_verify** is set, the sender is accepted with a warning message after a temporary verification failure.

Exim remembers temporary sender verification errors in a hints database. Subsequent temporary errors for the same address from the same host within 24 hours cause a 451 error after MAIL instead of after the data. This reduces the data on the reject log and also the amount repeatedly transferred over the net.

If **sender_verify_max_retry_rate** is set greater than zero, and the rate of temporary rejection of a specific incoming sender address from a specific host, in units of rejections per hour, exceeds it, the temporary error is converted into a permanent verification error. This should help in stopping hosts hammering too frequently with temporarily failing sender addresses. The default value of the option is 12, which means that a sender address that has a temporary verification error more than once every 5 minutes will soon get permanently rejected. Once permanent rejection has been triggered, subsequent temporary failures will all cause permanent errors, until there has been an interval of at least 24 hours since the last failure. After 24 hours, the hint expires.

What happens if verification fails with a permanent error depends on the setting of the **sender_verify_reject** option. If it is set (the default) then the message is rejected. Otherwise a warning message is logged, and processing continues.

Because remote postmasters always want to see the message headers when there is a problem, Exim does not give an error response immediately a sender address fails, but instead it reads the data for the message first. The headers of rejected messages are written to the reject log, for use in tracking down the problem or tracing mail abusers. Up to three envelope recipients are also logged with the headers.

Unfortunately, there are a number of mailers in use that treat any SMTP error response given after the data has been transmitted as a temporary failure. Exim sends code 550 when it rejects a message because of a bad sender, and RFC 821 is quite clear in stating that all codes starting with 5 are always 'permanent negative completion' replies. However, it does not give any guidance as to what should be done on receiving such replies, and some mailers persist in trying to send messages when they receive such a code at the end of the data.

To get round this, Exim keeps a database in which it remembers the bad sender address and host name when it rejects a message. If the same host sends the same bad sender address within 24 hours, Exim rejects the message at the MAIL command, before it reads the data for the message. This should prevent the sender from trying to send the message again, but there seem to be plenty of broken mailers out there that do keep on trying, sometimes for days on end.

In an attempt to shut such MTAs up, if the same host sends the same bad sender for a third time within 24 hours, MAIL is accepted, but all subsequent RCPT commands are rejected with a 550 error code. This means 'unknown user' and if a remote mailer doesn't treat that as a hard error, it is very seriously broken.

The **sender_verify_hosts** option can be used to restrict hosts and RFC 1413 idents for which sender verification is not applied. If a cluster of hosts all check incoming external messages, there is no need to waste effort checking mail sent between them. For example:

```
  sender_verify_hosts = "! *.ref.book : ! exim@mailer.fict.book"
```

39.3 Fixing bad senders

It is unfortunately the case that lots of messages are sent out onto the Internet with invalid senders. In some cases, the message itself contains a valid return address in one of its headers. If the **sender_verify_fixup** option is set as well as **sender_verify**, Exim does not reject a message if the sender is invalid, provided it can find a **Sender:**, **Reply-to:**, or **From:** header containing a valid address. Instead, it replaces the sender with the valid address, and records the fact that it has done so by adding a header of the form:

```
  X-BadReturnPath: <invalid address> rewritten using <name> header
```

If there are several occurrences of any of the relevant headers, they are all checked. If any **Resent-** headers exist, it is those headers that are checked rather than the original ones.

The fixup happens for both permanent and temporary errors. This covers the case when the bad addresses refer to some DNS zone whose nameservers are unreachable. This approach is, of course, fixing the symptom and not the disease.

If **sender_verify_fixup** is set when **sender_verify_reject** is false, Exim does not modify the message, but records in the log the fixup it would have made.

39.4 Header verification

Exim's sender verification options can be used to block messages with bad envelope senders. However, if a message arrives with a null envelope sender, that is, if the SMTP command was

```
MAIL FROM:<>
```

then Exim has nothing to check, and is forced to accept the message (unless it fails another check, of course). If **headers_sender_verify_errmsg** is set, then for messages that have null senders (purporting to be mail delivery error messages), Exim does some checking of the RFC 822 headers. It looks for a valid address in the **Sender:**, **Reply-to:**, and **From:** headers, and if one cannot be found, the message is rejected, unless **headers_checks_fail** is false, in which case it just makes a warning entry in the reject log.

If there are several occurrences of any of the relevant headers, they are all checked. If any **Resent-** headers exist, it is those headers that are checked rather than the original ones.

Unfortunately, because it has to read the message before doing this check, the rejection happens after the end of the data, and it is known that some mailers do not treat hard (5*xx*) errors correctly at this point – they keep the message on their spools and try again later, but that is their problem, though it does waste some resources.

The option **headers_sender_verify** is also available. It insists on there being a valid **Sender:**, **Reply-to:**, or **From:** header on all incoming SMTP messages, not just those with null senders.

The **sender_verify_hosts** option applies to both of these header checking options as well as to checks on envelope senders (**sender_verify**).

A common spamming ploy is to send syntactically invalid headers such as

```
To: @
```

The option **headers_check_syntax** causes Exim to check the syntax of all headers that can contain lists of addresses (**Sender:**, **From:**, **Reply-to:**, **To:**, **Cc:**, and **Bcc:**) on all incoming messages (both local and SMTP). This is a syntax check only. Like the **headers_sender_verify** options, the rejection happens after the end of the data, and it is also controlled by **headers_checks_fail**; if that is false, a bad message is accepted, with a warning in the reject log.

39.5 Receiver verification

By default, Exim just checks the syntax of addresses given in the SMTP RCPT command. This minimizes the time required for an SMTP message transfer, and also makes it possible to provide special processing for unknown local parts in local domains, by using a **smartuser** director to pass messages with unknown local parts to a script or to another host.

Some installations prefer to check receiver addresses as they are received. If the **receiver_verify** option is set, the same code that is used by the **-bv** option is used to check incoming addresses from remote hosts that match **receiver_verify_hosts**, whose default setting is to match all hosts. If verification fails, a permanent negative response is given to the RCPT command. If there is a temporary failure, a temporary error is given, unless **receiver_try_verify** is set, in which case the address is accepted.

It is also possible to restrict the addresses that are verified to certain domains by setting **receiver_verify_addresses**, and receiver verification can also be made conditional on the sender address by setting **receiver_verify_senders**. All of these options operate only when **receiver_verify** or **receiver_try_verify** is set.

40. Other policy controls on incoming mail

Exim provides a number of facilities for controlling incoming mail from remote hosts, in addition to the verification options described in the previous chapter. These controls can be used to stop unwanted messages getting into your machine. After a message has been accepted, the filtering mechanism described in chapter 41 can be used to check it before going ahead with delivery.

An MTA is said to *relay* a message if it receives it from some host and delivers it directly to another host as a result of a remote address contained within it. Expanding a local address via an alias or forward file and then passing the message on to a remote host does not count as relaying. There are special options for controlling which remote hosts may use the local host as a relay.

The options described in this chapter control three stages of checking that are applied to an incoming SMTP message:

(1) At the start of an SMTP connection, a check on the remote host is made, leading to one of the following conclusions:

   (i) No mail whatsoever is acceptable from the remote host.

   (ii) The remote host is permitted to send messages to local recipients only, but is not permitted to use the local host as a relay.

   (iii) The remote host is permitted to send messages to local recipients, and can also use the local host as a relay to certain specified domains only.

   (iv) The remote host is permitted to send mail to any recipient.

   If the host is completely unacceptable, the SMTP connection may be rejected immediately, or (depending on the configuration) the message may be refused later on by a rejection at the end of the message (so the headers can be logged) or by rejecting every recipient.

(2) The message's sender, which is obtained from the MAIL command, is checked. Again there is a choice of immediate rejection, or delayed rejection of all recipients.

(3) Unless there are no controls on relaying, the recipient address in each RCPT command is checked.

These checks are all in addition to any verification that may be enabled. The following sections give details of the various checking options. The **-bh** command line option can be used to run fake SMTP sessions for the purpose of testing them.

40.1 Host checking using RBL

The Realtime Blocking List (RBL) is a blacklist of hosts that is maintained in the DNS. See **http://maps.vix.com/rbl/** for the background to this. Since the RBL was created, a number of other similar lists (DUL, ORCA, IMRSS) have sprung up. These all operate in the same way. If the **rbl_domains** option is set, Exim looks up inverted incoming IP addresses in each of the given domains, provided the remote host matches **rbl_hosts** (whose default is to match all hosts). For example, if the setting is

```
rbl_domains = rbl.maps.vix.com:dul.maps.vix.com
```

and an SMTP call is received from the host whose IP address is 131.111.8.1, then DNS lookups for address records for

```
1.8.111.131.rbl.maps.vix.com
```
and
```
1.8.111.131.dul.maps.vix.com
```

are done. Each domain in **rbl_domains** can be followed by '/warn' or '/reject' to specify what is to be done when a match is found, for example:

```
rbl_domains = rbl.maps.vix.com/warn : dul.maps.vix.com/reject
```

The action for domains without either of these is controlled by **rbl_reject_recipients**, which implies '/reject' when set.

Warning consists of writing a message to the main and reject logs, and, if **rbl_warn_header** is true (the default), adding an **X-RBL-Warning:** header to the message. This can be detected later by system or user filter files. If a host appears in several RBL lists, more than one such header may be added to a message.

Rejection is done by refusing all recipients, that is, by giving permanent error returns to all RCPT commands, except for any recipients that are listed in **recipients_reject_except**. It is fairly common to set

```
  recipients_reject_except = postmaster@your.domain
```

to allow your host to accept mail to the postmaster from blacklisted hosts. If a TXT record associated with the host is found in the RBL domain, its contents are returned as part of the 550 rejection message, unless **prohibition_message** is set (see section 40.6), in which case a locally-specified message (possibly including the TXT data) is used. If a lookup times out or otherwise fails to give a decisive answer, the mail is not blocked.

40.2 Other host checking

Exim rejects incoming SMTP calls from any host that matches **host_reject**. For example:

```
  host_reject = ! xxx.yy.zz : *.yy.zz : ! *.zz
```

rejects mail from any host outside the **zz** domain, and all hosts in the **yy.zz** domain, except for **xx.yy.zz**. The use of wildcarded names implies a reverse DNS lookup of the incoming IP address. This can be avoided by using IP addresses. See section 7.16 for details.

Calls are rejected as a result of these options by sending a 5*xx* error code as soon as the connection is received. Since this does not relate to any particular message, the remote host is likely to keep on trying to send mail (possibly to an alternative MX host) until it times out. This may be what is wanted in some circumstances (for example, you want temporarily to hold back all incoming mail from some host), but when dealing with incoming spam, for example, one normally wants messages to be rejected once and for all, and in thist case, **host_reject_recipients** should be used instead of **host_reject**.

A call from a host which matches **host_reject_recipients** is not rejected at the start; instead, every RCPT command is subsequently rejected, which should cause the remote MTA to cease trying to deliver the message. This style of blocking also has the advantage of catering for exceptions for certain recipients, via the **recipients_reject_except** option. This is commonly set to the local postmaster address.

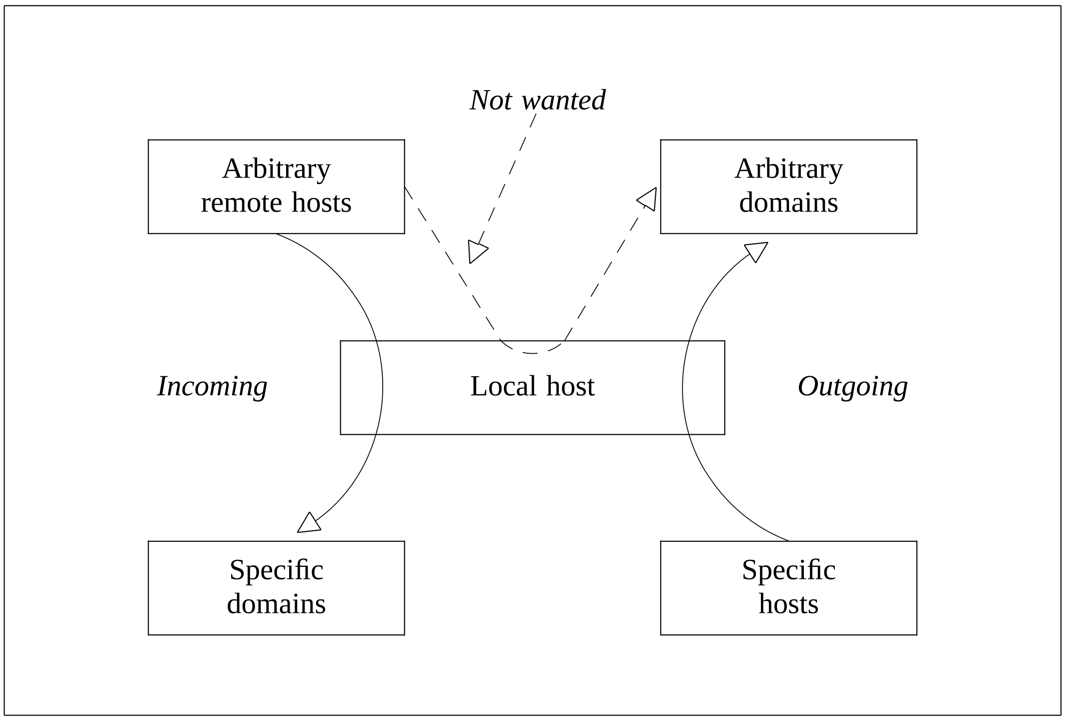
40.3 Sender checking

Incoming messages can be rejected on the basis of the sender address, as given in the MAIL command. A list of senders to reject is set by the **sender_reject** configuration option; see its description in chapter 11 for details.

Some MTAs continue to try to deliver a message even after receiving a 5*xx* error code for MAIL. The alternative configuration option **sender_reject_recipients** is provided for use in such cases. It accepts the MAIL command but rejects all subsequent RCPT commands.

40.4 Control of relaying

There are two aspects of control over relaying via the local host, which might be termed 'incoming' and 'outgoing'. A host which is acting as a gateway or an MX backup is concerned with incoming relaying from arbitrary hosts to a specific set of domains. A host which is acting as a smart host for a number of clients is concerned with outgoing relaying from those clients to the Internet at large. Often the same host is fulfilling both functions, as illustrated in the diagram below, but in principle these two kinds of relaying are entirely independent, and are therefore controlled by two separate options. What is not wanted is the transmission of mail from arbitrary remote hosts through your system to arbitrary domains.



Controlled relaying

Incoming relaying is controlled by restricting the domains to which an arbitrary host may send; outgoing relaying is controlled by restricting the hosts which may send to an arbitrary domain. If an arbitrary host can send via the local host to an arbitrary domain, the host is open to abuse.

The relaying check happens whenever a message's recipient is received, that is, immediately after a RCPT command. The first check is whether the address would cause relaying at all: if its domain matches something in **local_domains** then it is destined to be handled on the local host as a local address, and relaying is not involved, unless the 'percent hack' is in use. In this case, the local part is converted into a new address and that is then checked.

When the relevant domain is not in **local_domains**, there is first a check for legitimate incoming relaying, by seeing if it matches **relay_domains**, or, when **relay_domains_include_local_mx** is set, if it is a domain with an MX record pointing to the local host. If it does match, this is an acceptable incoming relay, and it is permitted to proceed.

For example, if the FooBar company has a firewall machine through which all mail from external hosts must pass, and this machine's configuration contains

```
local_domains = foobar.com
relay_domains = *.foobar.com
```

then mail from external hosts is rejected, unless it is for a domain ending in **foobar.com**.

If a recipient address is neither for a local domain nor an incoming relay, it must be an outgoing relay, and it is accepted only if the sending host is permitted to relay to arbitrary domains, and if the sender address is acceptable. The set of hosts is specified by **host_accept_relay**. For example, if the FooBar company's IP network is 192.153.213.0, and all hosts on that network send their outgoing mail via the firewall machine, then its configuration should contain

```
host_accept_relay = 192.153.213.0/24
```

in order to allow only the internal hosts to use it as a relay to arbitrary domains. Exim does not make an exception for the loopback IP address, so if you want to permit relaying from processes on the local host using this method, you need to set

```
host_accept_relay = 127.0.0.1
```

In addition to the test on the host, if **sender_address_relay** is set, the sender's address from the MAIL command must match one of its patterns to allow outgoing relaying to an arbitrary domain. Also, if there are any rewriting rules with the 'X' flag set, such an address is rewritten using those rules, and the result (if different) must verify successfully. See section 32.8 for an example of how this can be used.

Normally, therefore, both the host and the sender must be acceptable before an outgoing relay is allowed to proceed. However, if **relay_match_host_or_sender** is set, an address is accepted for outgoing relaying if *either* the host *or* the sender is acceptable. Of course, sender addresses can easily be forged, but the sender check does mean you can prevent some kinds of unwanted mail from going through your host.

Both **relay_domains** and **host_accept_relay** are unset by default, which means that no relaying of any kind is enabled. This does not prevent a local user from setting up forwarding to some external system, but it does prevent the 'percent hack' from working even when **percent_hack_domains** is set.

If you have a list of domains that any host can relay to, but there are no hosts that are permitted to relay to arbitrary domains (for example, if your host is an MX backup for some domains), then set **relay_domains**.

If the recipient address is an RFC 821 source routed address, that is, an address of the form **<@hop1,@hop2:user@domain>**, it is the final domain which is tested. By default, however, Exim will send the message to the **hop1** domain, unless it is a local domain. The **collapse_source_routes** option can be used to prevent this.

As all the relay checking is done at RCPT time on incoming messages, the directors and routers are not involved. Depending on the configuration of these drivers, an address that appears to be remote to the relay checking code (that is, its domain does not match **local_domains**) may nevertheless end up being delivered locally, and similarly an apparently local address may end up being delivered to some other host.

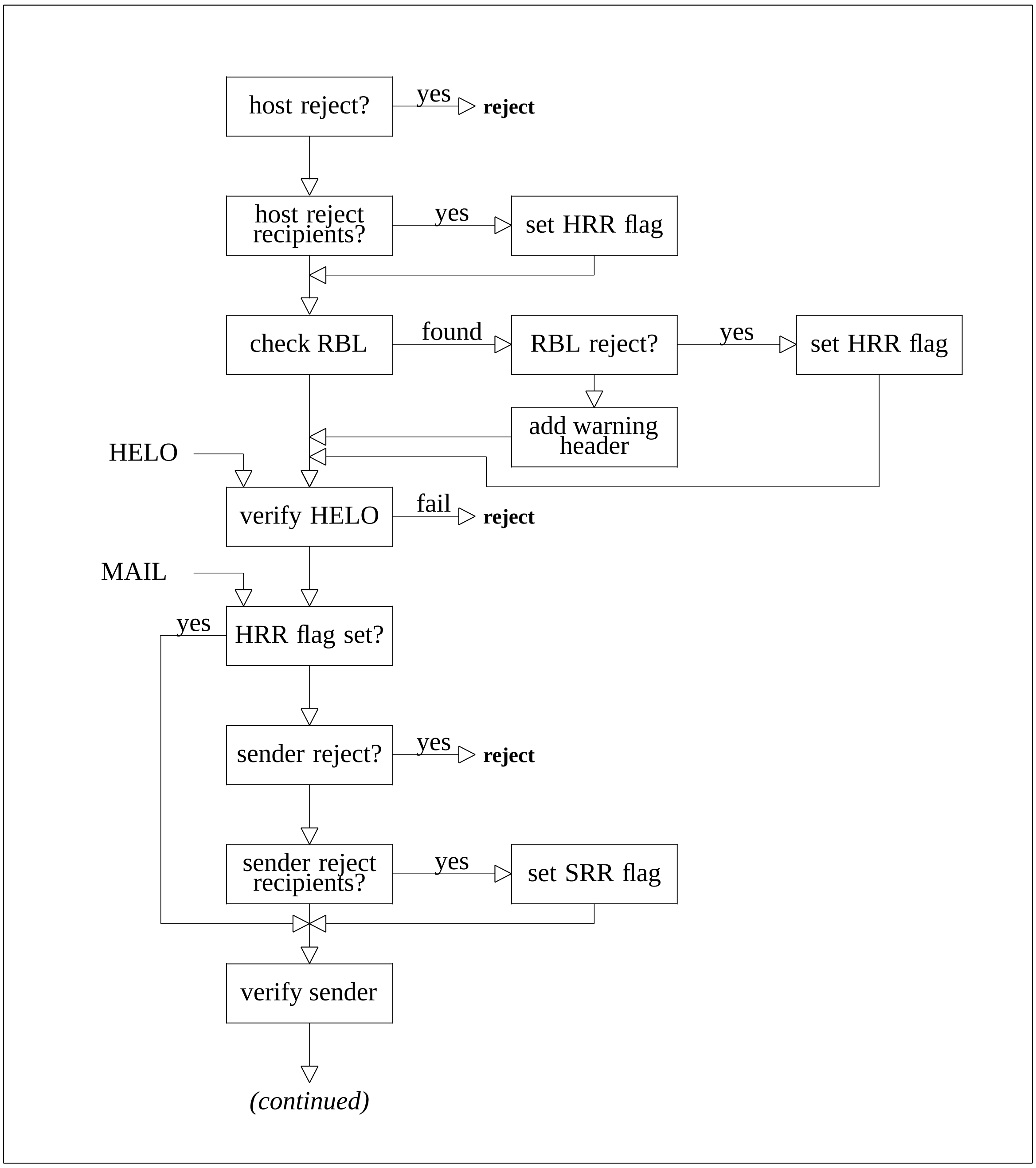
None of the relay checking applies when mail is passed to Exim locally using the **-bm**, **-bs** or **-bS** options, but it does apply when **-bs** is used from **inetd**.

Exim does not attempt to fully qualify domains at RCPT time. If an incoming message contains a domain which is not fully qualified, it is treated as a non-local, non-relay domain (unless partial domains are included in **local_domains** or **relay_domains**, but this is not recommended). The use of domains that are not fully qualified is non-standard, but it is a commonly encountered usage when an MTA is being used as a smart host by some remote UA. In this situation, it would be usual to permit the UA host to relay to any domain, so in practice there is not normally a problem.

40.5 Policy checking flowchart

The diagrams below shows how the various policy checks are applied to an incoming message from a remote host. The normal flow of control is vertically down the left-hand set of boxes.

If verification of a sender fails, rejection may be immediate, or it may follow later after the RCPT command or after the data has been received (see section 39.2). If recipient verification fails, rejection is immediate.



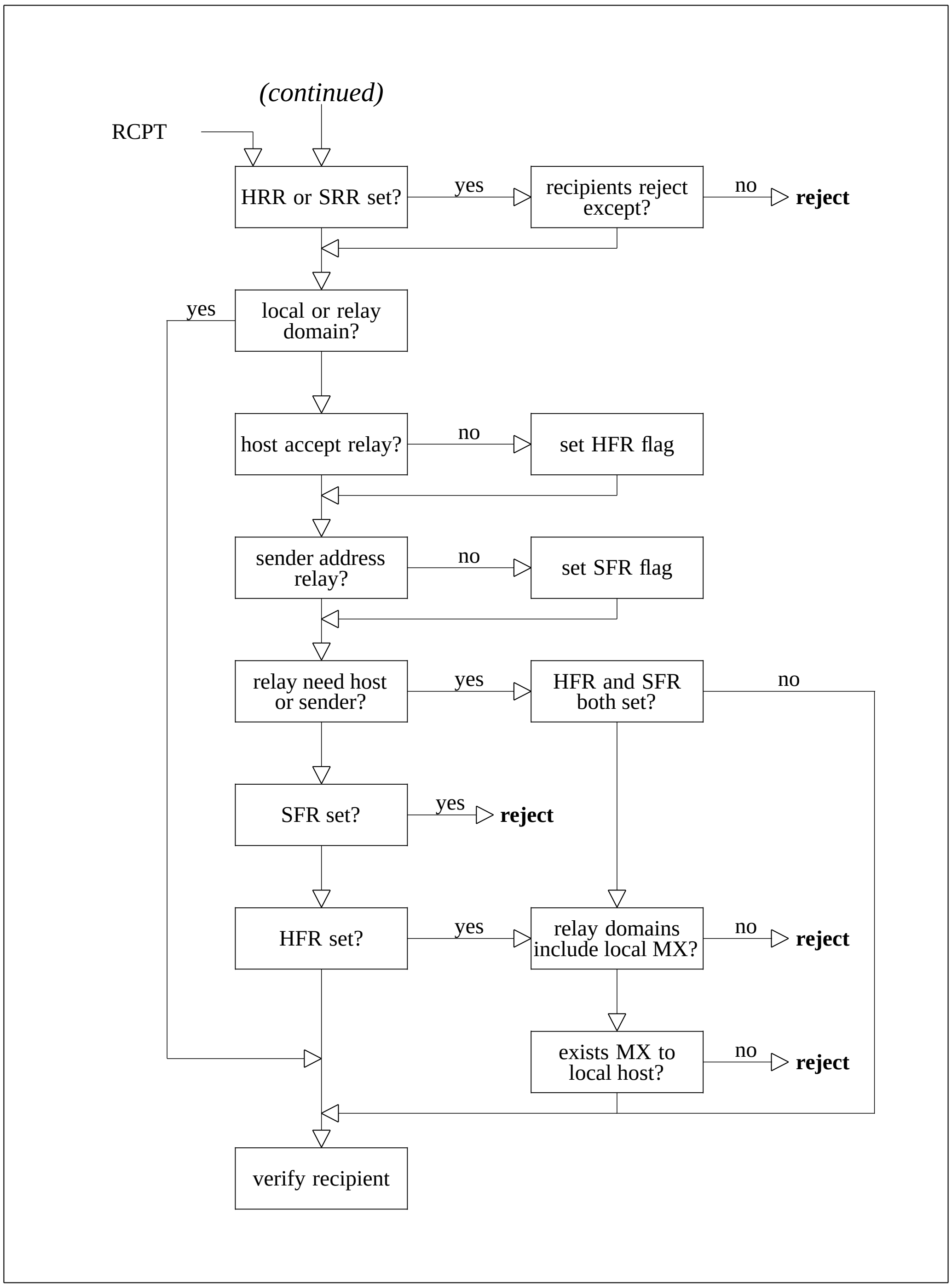
Policy checking, part 1

40.6 Customizing pr ohibition messages

It is possible to add a site-specific message to the error response that is sent when an incoming SMTP command fails for policy reasons, for example if the sending host is in a host reject list. This is done by setting the option **prohibition_message**, which causes one or more additional response lines with the same error code and a multiline marker to be output before the standard response line. For example, setting

```
  prohibition_message = "contact postmaster@my.site for details"
```

causes the response to a RCPT command for a blocked recipient to be



Policy checking, part 2

```
550-contact postmaster@my.site for details
550 rejected: administrative prohibition
```

The string is expanded, and so it can do file lookups if necessary. If it ends up as an empty string, no additional response is transmitted. To make it possible to distinguish between the several different types of administrative rejection, the variable **$prohibition_reason** is set to a characteristic text string in each case. The possibilities are as follows:

`host_accept_relay`	the host is not in an **accept_relay** list
`host_reject`	the host is in a reject list
`host_reject_recipients`	the host is in a **reject_recipients** list
`rbl_reject`	the host is rejected by an RBL domain
`sender_relay`	the sender is not in a sender relay list
`sender_reject`	the sender is in a reject list
`sender_reject_recipients`	the sender is in a **reject_recipients** list
`sender_verify`	sender verification failed

In addition, if **relay_match_host_or_sender** is set, there is

`sender+host_accept_relay`	the sender is not in a sender relay list *and* the host is not in an accept relay list

For example, if the configuration contains

```
prohibition_message = "${lookup{$prohibition_reason}lsearch\
  {/etc/exim/reject.messages}{$value}}"
```

and the file **/etc/exim/reject.messages** contains (*inter alia*)

```
host_accept_relay:  host not in relay list
```

then a response to a relay attempt might be

```
550-host not in relay list
550 relaying to <santa@northpole.com> prohibited by administrator
```

Because some administrators may want to put in quite long messages, and it isn’t possible to get newlines into the text returned from an lsearch lookup, Exim treats the vertical bar character as a line separator in this text. If you want the looked up text to be re-expanded, you can use the **expand** operator. For example, the setting

```
prohibition_message = "${lookup{$prohibition_reason}lsearch\
  {/etc/exim/reject.messages}{${expand:$value}}}"
```

when used with a file entry of the form

```
host_accept_relay:  Host $sender_fullhost is not permitted to
                    relay |through $primary_hostname.
```

might produce

```
550-Host that.host.name [111.222.3.4] is not permitted to relay
550-through this.host.name.
550 relaying to <penguins@southpole.com> prohibited by administrator
```

When the prohibition is due to an entry in a Realtime Blocking List and a message is available from a DNS TXT record, that text is available in the **$rbl_text** variable. If **prohibition_message** is not set, then the TXT data is always included in the rejection message.

41. System-wide message filtering

The previous chapters describe checks that can be applied to messages before they are accepted by a host. There are also mechanisms for checking messages once they have been received, but before they are delivered. A *system message filter* can be run each time a delivery process is started for a message. It is also possible to run a centrally-defined filter file once for each local address, as part of the directing for that address.

41.1 The system message filter

The system message filter operates in a similar manner to users' filter files, but it is run just once per message (however many recipients is has) at the start of a delivery attempt, before any routing or directing is done. If a message fails to be completely delivered at the first attempt, the filter is run again at the start of the every retry.

There are two special conditions which, though available in users' filter files, are designed for use in system filters. The condition **first_delivery** is true only for the first attempt at delivering a message, while **manually_thawed** is true only if the message has been frozen, and subsequently thawed by an admin user. An explicit forced delivery counts as a manual thaw, but thawing as a result of the **auto_thaw** setting does not.

If the filter sets up any deliveries of its own, an extra header line is added to them with the name **X-Envelope-to:**. This contains up to 100 of the original message's envelope recipients. If the filter specifies any significant deliveries, then the message's own recipient list is ignored; otherwise it is added to any recipients set up by the filter.

The **message_filter** option names the filter file, while **message_filter_user** and **message_filter_group** specify the uid and gid to be used while processing it. If they are not set, then the exim uid is used if available and if **seteuid()** is available; otherwise root is used. There are also options for specifying which transports are to be used if the filter generates any file, pipe or autoreply deliveries.

The filter file can contain any of the normal filtering commands, as described in the separate document *Exim's User interface to mail filtering*. However, because the system filter is run just once per delivery attempt, the variable **$local_part** is not available, nor does the 'personal' condition make any sense.

The filter variables **$n0** – **$n9** can be used in a system filter; when it finishes, their values are copied into **$sn0** – **$sn9** and are thereby made available to users' filter files. Thus a system filter can, for example, set up a 'score' for a message to which users' filter files can refer.

In addition to the filter commands available in user's files there are some extra commands which are available only in system filter files:

```
fail
freeze
headers add <string>
headers remove <string>
```

As well as the additional commands, there is also an extra expansion variable, **$recipients**, containing a list of all the recipients of the message, separated by commas and white space. The extra commands and variable are not available in ordinary users' filter files. They are faulted in normal use and in testing via **-bf**, but not if **-bF** is used.

The **freeze** and **fail** commands can optionally be followed by the word **text** and a string containing an error message, for example:

```
fail text "this message looks like spam to me"
```

If either **freeze** or **fail** is obeyed in a system filter file, no deliveries are done, not even those set up by **mail** commands in the filter. See the **freeze_tell_mailmaster** option for a way of having a message sent when a message is frozen.

The keyword **text** is optional if the next character is a double quote. The **fail** command causes all recipients to be failed, while **freeze** suspends all delivery attempts. It is ignored if the message has been manually unfrozen and not manually frozen since. This means that automatic freezing by a system filter can be used as a way of checking out suspicious messages. If a message is found to be all right, manually unfreezing it allows it to be delivered.

The argument for the **headers add** is a string which is expanded and then added to the end of the message's headers. It is the responsibility of the filter maintainer to make sure it conforms to RFC 822 syntax. Leading white space is ignored, and if the string is otherwise empty, or if the expansion is forced to fail, the command has no effect. A newline is added at the end of the string if it lacks one. More than one header may be added in one command by including '\n' within the string.

The argument for **headers remove** is a colon-separated list of header names. This command applies only to those headers that are stored with the message; ones such as **Envelope-To:** and **Return-Path:** that are added at delivery time cannot be removed by this means.

Take great care with the **fail** command when basing the decision to fail on the contents of the message, because this option causes a normal delivery error message to be generated, and it will of course include the contents of the original message and will therefore trigger the **fail** command again (causing a mail loop) unless steps are taken to prevent this. Testing the **error_message** condition is one way to prevent this. You could use, for example

```
if
  $message_body contains "this is spam" and not error_message
then
  fail text "spam is not wanted here"
endif
```

though of course that might still let through unwanted messages. The alternative is clever checking of the body and/or headers to detect error messages caused by the filter.

41.2 Per-address filtering

In contrast to the system filter, which is run just once per message for each delivery attempt, it is also possible to set up a system-wide filtering operation that runs once for each address, for local addresses only. In this case, variables such as **$local_part** and **$domain** can be used, and indeed, the choice of filter file could be made dependent on them. This is an example of a director which implements such a filter:

```
central_filter:
  driver = forwardfile
  file = /central/filters/${local_part}
  no_check_local_user
  no_verify
  filter
  allow_system_actions
```

The setting of **allow_system_actions** permits the use of **freeze** and **fail** in the filter file, but not the **headers** command or the **$recipients** variable.

42. SMTP processing

Two kinds of SMTP processing are supported: SMTP over TCP/IP, and so-called ‘batched SMTP’. The latter is the name for a process in which batches of messages are stored in files, using SMTP commands as separators and to contain the envelope information. Such batches are delivered to or received from other systems using some transport mechanism other than Exim. For each kind of SMTP processing there are two aspects: outgoing and incoming.

42.1 Outgoing SMTP over TCP/IP

This is implemented by the **smtp** transport. If the greeting line from the remote host contains the string ‘ESMTP’, Exim sends an EHLO command instead of HELO, and if it is told that the SIZE parameter is supported, it adds SIZE=<*n*> to each subsequent MAIL command. The value of <*n*> is the message size plus the value of the **size_addition** option (default 1024) to allow for additions to the message such as per-transport header lines, or changes made in a transport filter. If **size_addition** is set negative, the use of SIZE is suppressed.

Responses from the remote host are supposed to be terminated by CR followed by LF. However, there are known to be hosts that do not send CR characters, so in order to be able to interwork with such hosts, Exim treats LF on its own as a line terminator.

If a message contains a number of different addresses, all those with the same characteristics (for example, the same envelope sender) that resolve to the same set of hosts, in the same order, are sent in a single SMTP transaction, even if they are for different domains, unless there are more than the setting of the **max_rcpts** option in the **smtp** transport allows, in which case they are split into groups containing no more than **max_rcpts** addresses each. If **remote_max_parallel** is greater than one, such groups may be sent in parallel sessions. The order of hosts with identical MX values is not significant when checking whether addresses can be batched in this way.

When the **smtp** transport suffers a temporary failure that is not message-related, Exim updates its transport-specific database, which contains records indexed by host name that remember which messages are waiting for each particular host. It also updates the retry database with new retry times. Exim’s retry hints are based on host name plus IP address, so if one address of a multi-homed host is broken, it will soon be skipped most of the time. See the next section for more detail about error handling.

When a message is successfully delivered over a TCP/IP SMTP connection, Exim looks in the hints database for the transport to see if there are any queued messages waiting for the host to which it is connected. If it finds one, it creates a new Exim process using the **-MC** option (which can only be used by a process running as root or the Exim user) and passes the TCP/IP socket to it. The new process does only those deliveries that are routed to the connected host, and may in turn pass the socket on to a third process, and so on. The **batch_max** option of the **smtp** transport can be used to limit the number of messages sent down a single connection. The second and subsequent messages delivered down an SMTP connection are identified in the main log by the addition of an asterisk after the closing square bracket of the IP address.

When a delivery process that is part of a queue run passes a socket on to a new delivery process, it writes the new process’ id to a file in Exim’s spool directory whose name is ‘qr’ followed by the process id of the queue runner. It also passes on the queue runner’s process id using the **-MCQ** option, in case the socket is passed on to yet another process. The queue running process reads the file when the original process ends, and waits for each process listed therein to finish before proceeding. Because each process writes the process id of its child into the file before it itself finishes, the end of the file is reached only when the entire tree of processes spawned by the original delivery process is complete.

42.2 Errors in outgoing SMTP

Three different kinds of error are recognized for outgoing SMTP: host errors, message errors, and recipient errors.

(1) A host error is not associated with a particular message or with a particular recipient of a message. The host errors are:

- Connection refused or timed out,
- Any error response code on connection,
- Any error response code to HELO or EHLO,
- Loss of connection at any time, except after '.',
- I/O errors at any time,
- Timeouts during the session, other than in response to MAIL, RCPT or the '.' at the end of the data.

A host error causes all addresses to be deferred, and retry data to be created for the host. It is not tried again, for any message, until its retry time arrives. If the current set of addresses are not all delivered in this run (to some alternative host), the message is added to the list of those waiting for this host, so if it is still undelivered when a subsequent successful delivery is made to the host, it will be sent down the same SMTP connection.

(2) A message error is associated with a particular message when sent to a particular host, but not with a particular recipient of the message. The message errors are:

- Any error response code to MAIL, DATA, or the '.' that terminates the data,
- Timeout after MAIL,
- Timeout or loss of connection after the '.' that terminates the data. A timeout after the DATA command itself is treated as a host error, as is loss of connection at any other time.

A permanent error response (5*xx*) causes all addresses to be failed, and a delivery error report to be returned to the sender. A temporary error response (4*xx*) or one of the timeouts causes all addresses to be deferred. Retry data is not created for the host, but instead, a retry record for the combination of host plus message id is created. The message is not added to the list of those waiting for this host. This ensures that the failing message will not be sent to this host again until the retry time arrives. However, other messages that are routed to the host are not affected, so if it is some property of the message that is causing the error, it will not stop the delivery of other mail.

If the remote host specified support for the SIZE parameter in its response to EHLO, then Exim adds SIZE=*nnn* to the MAIL command, so an over-large message will cause a message error because it will arrive as a response to MAIL.

(3) A recipient error is associated with a particular recipient of a message. The recipient errors are:

- Any error response to RCPT,
- Timeout after RCPT.

A permanent error response (5*xx*) causes the recipient address to be failed, and a delivery error report to be returned to the sender. A temporary error response (4*xx*) or a timeout causes the failing address to be deferred, and routing retry data to be created for it. This is used to delay processing of the address in subsequent queue runs, until its routing retry time arrives. This applies to all messages, but because it operates only in queue runs, one attempt will be made to deliver a new message to the failing address before the delay starts to operate. This ensures that, if the failure is really related to the message rather than the recipient ('message too big for this recipient' is a possible example), other messages have a chance of getting delivered. However, if a delivery to the address does succeed, the retry information gets cleared, so all stuck messages get tried again, and the retry clock is reset.

The message is not added to the list of those waiting for this host. Use of the host for other messages is unaffected, and except in the case of a timeout, other recipients are processed independently, and may be successfully delivered in the current SMTP session. After a timeout it is of course impossible to proceed with the session, so all addresses get deferred. However, those other than the one that failed do not suffer any subsequent retry delays. Therefore, if one recipient is causing trouble, the others have a chance of getting through when a subsequent delivery attempt occurs before the failing recipient's retry time.

In all cases, if there are other hosts (or IP addresses) available for the current set of addresses (for example, from multiple MX records), they are tried in this run for any undelivered addresses, subject of course to their own retry data. In other words, recipient error retry data does not take effect until the next delivery attempt.

Some hosts have been observed to give temporary error responses to every MAIL command at certain times ('insufficient space' has been seen). It would be nice if such circumstances could be recognized, and defer data for the host itself created, but this is not possible within the current Exim design. What actually happens is that retry data for every (host, message) combination is created.

The reason that timeouts after MAIL and RCPT are treated specially is that these can sometimes arise as a result of the remote host's verification procedures. Exim makes this assumption, and treats them as if a temporary error response had been received. A timeout after '.' is treated specially because it is known that some broken implementations fail to recognize the end of the message if the last character of the last line is a binary zero. Thus is it helpful to treat this case as a message error.

Timeouts at other times are treated as host errors, assuming a problem with the host, or the connection to it. If a timeout after MAIL, RCPT, or '.' is really a connection problem, the assumption is that at the next try the timeout is likely to occur at some other point in the dialogue, causing it then to be treated as a host error.

There is experimental evidence that some MTAs drop the connection after the terminating '.' if they don't like the contents of the message for some reason, in contravention of the RFC, which indicates that a 5*xx* response should be given. That is why Exim treats this case as a message rather than a host error, in order not to delay other messages to the same host.

42.3 Variable Envelope Return Paths (VERP)

Variable Envelope Return Paths – see **ftp://koobera.math.uic.edu/www/proto/verp.txt** – can be supported in Exim by using the **return_path** generic transport option to rewrite the return path at transport time. For example, the following could be used on an smtp transport:

```
return_path = "\
  ${if match {$return_path}{^(.+?)-request@your.domain\\$}\
  {$1-request=$local_part%$domain@your.domain}fail}"
```

This has the effect of rewriting the return path (envelope sender) on all outgoing SMTP messages, if the local part of the original return path ends in '-request', and the domain is **your.domain**. The rewriting inserts the local part and domain of the recipient into the return path. If, for example, a message with return path **somelist-request@your.domain** is sent to **subscriber@other.domain**, the return path is rewritten as

```
somelist-request=subscriber%other.domain@your.domain
```

For this to work, you must arrange for outgoing messages that have '-request' in their return paths to have just a single recipient. This can be done by setting

```
max_rcpt = 1
```

in the smtp transport. Otherwise a single copy of a message might be addressed to several different recipients in the same domain, in which case **$local_part** is not available (because it is not unique). Of course, if you do start sending out messages with this kind of return path, you must also configure Exim to accept the bounce messages that come back to those paths. Typically this would be done by setting a **suffix** option in a suitable director.

The overhead incurred in using VERP depends very much on the size of the message, the number of recipient addresses that resolve to the same remote host, and the speed of the connection over which the message is being sent. If a lot of addresses resolve to the same host and the connection is slow, sending a separate copy of the message for each address may take substantially longer than sending a single copy with many recipients (for which VERP cannot be used).

42.4 Incoming SMTP messages over TCP/IP

Incoming SMTP messages can be accepted in one of two ways: by running a listening daemon, or by using **inetd**. In the latter case, the entry in **/etc/inetd.conf** should be like this:

```
  smtp  stream  tcp  nowait  exim  /opt/exim/bin/exim  in.exim  -bs
```

Exim distinguishes between this case and the case of a user agent using the **-bs** option by checking whether the standard input is a socket using the **getpeername()** function.

By default, Exim does not make a log entry when a remote hosts connects or disconnects (either via the daemon or **inetd**), unless the disconnection is unexpected. It can be made to write such log entries by setting the **log_smtp_connections** option.

Commands from the remote host are supposed to be terminated by CR followed by LF. However, there are known to be hosts that do not send CR characters, so in order to be able to interwork with such hosts, Exim treats LF on its own as a line terminator.

The amount of disc space available is checked whenever SIZE is received on a MAIL command, independently of whether **message_size_limit** or **check_spool_space** is configured, unless **smtp_check_spool_space** is set false. A temporary error is given if there isn't enough. If **check_spool_space** is set, the check is for that amount of space plus the value given with SIZE, that is, it checks that the addition of the incoming message will not reduce the space below the threshold.

When a message is successfully received, Exim includes the local message id in its response to the final '.' that terminates the data. If the remote host logs this text it can help with tracing what has happened to a message.

The Exim daemon can limit the number of simultaneous incoming connections it is prepared to handle (see the **smtp_accept_max** option). It can also limit the number of simultaneous incoming connections from a single remote host (see the **smtp_accept_max_per_host** option). Additional connection attempts are rejected using the SMTP temporary error code 421.

On some operating systems the SIGCHLD signal that is used to detect when a subprocess has finished can get lost at busy times. However, the daemon looks for completed subprocesses every time it wakes up, so provided there are other things happening (new incoming calls, starts of queue runs), the completion of processes created to handle incoming calls should get noticed eventually. If, however, Exim appears not to be accepting as many incoming connections as expected, sending the daemon a SIGCHLD signal will wake it up and cause it to check for any completed subprocesses.

When running as a daemon, Exim can reserve some SMTP slots for specific hosts, and can also be set up to reject SMTP calls from non-reserved hosts at times of high system load – for details see the **smtp_accept_reserve**, **smtp_load_reserve**, and **smtp_reserve_hosts** options. The load check applies in both the daemon and **inetd** cases.

If neither **queue_only** nor the **-odq** command line option is set, Exim normally starts a delivery process for each message received. However, the number of simultaneously running delivery processes started in this way can be limited by the **smtp_accept_queue** and **smtp_accept_queue_per_connection** options, and the **queue_only_load** option can specify a system load average above which immediate delivery is suspended. When either limit is reached, subsequently received messages are just put on the input queue.

The controls that involve counts of incoming SMTP calls (**smtp_accept_max smtp_accept_queue**, **smtp_accept_reserve**) are not available when Exim is started up from the **inetd** daemon, since each connection is handled by an entirely independent Exim process. Control by load average is, however, available with **inetd**.

Exim can be configured to verify addresses in incoming SMTP commands as they are received. See chapter 39 for details. It can also be configured to rewrite addresses at this time – before any syntax checking is done. See section 32.6.

42.5 The VRFY, EXPN, and DEBUG commands

The SMTP command VRFY is accepted only when the configuration option **smtp_verify** is set, and if so, it runs exactly the same code as when Exim is called with the **-bv** option.

The SMTP command EXPN is is permitted only if the calling host matches **smtp_expn_hosts** (add 'localhost' if you want calls to 127.0.0.1 to be able to use it). A single-level expansion of the address is done. EXPN is treated as an 'address test' (similar to the **-bt** option) rather than a verification (the **-bv** option). If an unqualified local part is given as the argument to EXPN, it is qualified with **qualify_domain**.

Rejections of VRFY and EXPN commands are logged on the main and reject logs, and VRFY verification failures are logged on the main log for consistency with RCPT failures.

The SMTP command DEBUG is not supported at all. Occurrences of this command are rejected, and the incident is logged.

42.6 The ETRN command

RFC 1985 describes an SMTP command called ETRN that is designed to overcome the security problems of the TURN command (which has fallen into disuse). Exim recognizes ETRN if the calling host matches **smtp_etrn_hosts**. Attempts to use ETRN from other hosts are logged on the main and reject logs; when ETRN is accepted, it is logged on the main log.

The ETRN command is concerned with 'releasing' messages that are awaiting delivery to certain hosts. As Exim does not organize its message queue by host, the only form of ETRN that is internally supported is the one where the text starts with the '#' prefix, in which case the remainder of the text is specific to the SMTP server. A valid ETRN command causes a run of Exim with the **-R** option to happen, with the remainder of the ETRN text as its argument. For example,

```
ETRN #brigadoon
```

causes a delivery attempt on all messages with undelivered addresses containing the text 'brigadoon'.

For more control over what ETRN does, the **smtp_etrn_command** option can used. This specifies a command that is run whenever ETRN is received. For example:

```
smtp_etrn_command = /etc/etrn_command $domain $sender_host_address
```

The string is split up into arguments which are independently expanded. The expansion variable **$domain** is set to the argument of the ETRN command, and no syntax checking is done on the contents of this argument. A new freestanding process is created to run the command. Exim does not wait for it to complete, so its status code is not checked. As Exim is normally running under its own uid and gid when receiving incoming SMTP, it is not possible for it to change them before running the command.

42.7 Outgoing batched SMTP

Both the **appendfile** and **pipe** transports can be used for handling batched SMTP. Each has an option called **bsmtp** which, if set to anything other than 'none' causes the message to be output in SMTP format. The message is written to the file or pipe preceded by the SMTP commands MAIL and RCPT, and followed by a line containing a single dot. The SMTP command HELO is not normally used, but if the transport's **bsmtp_helo** option is set true, a HELO command line precedes each message.

Lines in the message starting with a dot get an extra dot added. If the **prefix** option is set, its contents are included after the SMTP commands, and the contents of **suffix** appear at the end of the message, before the terminating dot; normally these options are specified as empty, to override the defaults.

The value of the **bsmtp** option determines how multiple addresses in a single message may be batched, if other conditions permit. If the value of **bsmtp** is ‘one’, then there is no batching, and a copy of the message is output for each address. If the value is ‘domain’ then a single copy (with multiple RCPT commands) is output for all addresses that have the same domain. If the value is ‘all’ then only a single copy of the message is written. The batching is further constrained by other parameters:

- If any of the transport’s expandable strings contain a reference to **$local_part**, then no batching takes place.
- If any of the transport’s expandable strings contains a reference to **$domain**, then only domain batching is done.
- Addresses are not batched if they have different error addresses, associated hosts, header additions or removals and so on.
- The uid and gid for delivery must be explicitly set. This is normally done in the transport, but if they are specified by a router or director, batching occurs only for addresses that have the same uid/gid set up.

Here is an example of a transport and router for batched SMTP:

```
# transport
smtp_appendfile:
  driver = appendfile
  directory = /var/bsmtp/${host}
  bsmtp = all
  prefix =
  suffix =
  user = exim

# router
route_append:
  driver = domainlist
  transport = smtp_appendfile
  route_list = "some.domain  batch.host"
```

This causes messages addressed to **some.domain** to be written in batched SMTP format to **/var/bsmtp/batch.host**, with only a single copy of each message. Note that prefix and suffix must be explicitly changed from their defaults.

42.8 Incoming batched SMTP

The **-bS** command line option causes Exim to accept one or more messages by reading SMTP on the standard input, but to generate no responses. If the caller is trusted, then the senders in the MAIL commands are believed; otherwise the sender is always the caller of Exim. Unqualified senders and receivers are not rejected (there seems little point) but instead just get qualified. If **sender_verify** is set, sender verification takes place only if **sender_verify_batch** is set (it defaults unset). Receiver verification and administrative rejection is not done, even if configured. HELO and EHLO act as RSET; VRFY, EXPN, ETRN, HELP, and DEBUG act as NOOP; QUIT quits.

If any error is detected while reading a message, including a missing ‘.’ at the end, Exim gives up immediately. It writes details of the error to the standard output in a stylized way that the calling program should be able to make some use of automatically, for example:

```
554 Unexpected end of file
Transaction started in line 10
Error detected in line 14
```

It writes a more verbose version, for human consumption, to the standard error file, for example:

```
  An error was detected while processing a file of BSMTP input.
  The error message was:

    501 ’>’ missing at end of address

  The SMTP transaction started in line 10.
  The error was detected in line 12.
  The SMTP command at fault was:

     rcpt to:<malformed@in.com.plete

  1 previous message was successfully processed.
  The rest of the batch was abandoned.
```

The return code from Exim is zero only if there were no errors. It is 1 if some messages were accepted before an error was detected, and 2 if no messages were accepted.

43. Message processing

Exim performs various transformations on the sender and recipient addresses of all messages that it handles, and also on the messages' header lines. Some of these are optional and configurable, while others always take place. All of this processing, except rewriting as a result of routing, happens when a message is received, before it is first written to the spool.

RFC 822 makes provision for headers starting with the string **Resent-**. It states that in general, the **Resent-** fields should be treated as containing a set of information that is independent of the set of original fields, and that information for one set should not automatically be taken from the other. If Exim finds any **Resent-** headers in the message, it applies the header transformations described below only to the **Resent-** header set, leaving the unqualified set alone.

43.1 Unqualified addresses

By default, Exim expects every address it receives from an external host to be fully qualified. Unqualified addresses cause negative responses to SMTP commands. However, because SMTP is used as a means of transporting messages from MUAs running on personal workstations, there is sometimes a requirement to accept unqualified addresses from specific hosts or IP networks.

Exim has two options that separately control which hosts may send unqualified sender or receiver addresses in SMTP commands, namely **sender_unqualified_hosts** and **receiver_unqualified_hosts**. In both cases, if an unqualified address is accepted, it is qualified by adding the value of **qualify_domain** or **qualify_receiver**, as appropriate.

Any addresses that are unqualified are made fully qualified by adding **qualify_domain** or **qualify_ recipient**, as appropriate. Unqualified addresses are accepted only from local sources or from hosts that match one of the **receiver_unqualified** or **sender_unqualified** options, as appropriate.

43.2 The UUCP From line

Messages that have come from UUCP (and some other applications) often begin with a line containing the envelope sender and a timestamp, following the word 'From'. Examples of two common formats are:

```
From a.oakley@berlin.mus Fri Jan  5 12:35 GMT 1996
From f.butler@berlin.mus Fri, 7 Jan 97 14:00:00 GMT
```

This line precedes the RFC 822 header lines. For compatibility with Sendmail, Exim recognizes such lines at the start of messages that are submitted to it via the command line (that is, on the standard input). It does not recognize such lines in incoming SMTP messages, unless the sending host matches **ignore_fromline_hosts** or the **-bs** option was used for a local message and **ignore_fromline_local** is set. The recognition is controlled by a regular expression that is defined by the **uucp_from_pattern** option, whose default value matches the two common cases shown above and puts the address that follows 'From' into **$1**.

When the caller of Exim for a non-SMTP message is a trusted user, the message's sender address is constructed by expanding the contents of **uucp_sender_address**, whose default value is '$1'. This is then parsed as an RFC 822 address. If there is no domain, the local part is qualified with **qualify_domain** unless it is the empty string. However, if the command line **-f** option is used, it overrides the 'From' line.

If the caller of Exim is not trusted, the 'From' line is recognized, but the sender address is not changed. This is also the case for incoming SMTP messages that are permitted to contain 'From' lines.

Only one 'From' line is recognized. If there is more than one, the second is treated as a data line that starts the body of the message, as it is not valid as a header line. This also happens if a 'From' line is present in an incoming SMTP message from a source that is not permitted to send them.

43.3 The Bcc: header

If Exim is called with the **-t** option, to take recipient addresses from a message's headers, it removes any **Bcc:** header that may exist (after extracting its addresses), unless the message has no **To:** or **Cc:** header, in which case a **Bcc:** header with no addresses is left in the message, in accordance with RFC 822. If **-t** is not present on the command line, any existing **Bcc:** header is not removed.

If Exim is called to receive a message with the recipient addresses given on the command line, and there is no **Bcc:**, **To:**, or **Cc:** header in the message, it normally adds a **To:** header, listing the recipients. Some mailing list software is known to submit messages in this way, and in this case the creation of a **To:** header is not what is wanted. If the **always_bcc** option is set, Exim adds an empty **Bcc:** header instead in this circumstance.

43.4 The Date: header

If a message has no **Date:** header, Exim adds one, giving the current date and time.

43.5 The Delivery-date: header

Delivery-date: headers are not part of the standard RFC 822 header set. Exim can be configured to add them to the final delivery of messages. (See the generic **delivery_date_add** transport option.) They should not be present in messages in transit. If the **delivery_date_remove** configuration option is set (the default), Exim removes **Delivery-date:** headers from incoming messages.

43.6 The Envelope-to: header

Envelope-to: headers are not part of the standard RFC 822 header set. Exim can be configured to add them to the final delivery of messages. (See the generic **envelope_to_add** transport option.) They should not be present in messages in transit. If the **envelope_to_remove** configuration option is set (the default), Exim removes **Envelope-to:** headers from incoming messages.

43.7 The From: header

If an incoming message does not contain a **From:** header, Exim adds one containing the sender's address. This is obtained from the message's envelope in the case of remote messages; for locally-generated messages the calling user's login name and full name are used to construct an address, as described in section 43.14. They are obtained from the password file entry by calling **getpwuid()** (but see the **unknown_login** configuration option). The address is qualified with **qualify_domain**.

For compatibility with Sendmail, if an incoming, non-SMTP message has a **From:** header containing just the unqualified login name of the calling user, this is replaced by an address containing the user's login name and full name as described in section 43.14.

43.8 The Message-id: header

If an incoming message does not contain a **Message-id:** header, Exim constructs one and adds it to the message. The id is constructed from Exim's internal message id, preceded by the letter E to ensure it starts with a letter, and followed by @ and the primary host name. Additional information can be included in this header by setting the **message_id_header_text** option.

43.9 The Received: header

A **Received:** header is added at the start of every message. The contents of this header are defined by the **received_header_text** configuration option, and Exim automatically adds a semicolon and a timestamp to the configured string.

43.10 The Return-path: header

Return-path: headers are defined as something the MTA may insert when it does the final delivery of messages. (See the generic **return_path_add** transport option.) Therefore, they should not be present in messages in transit. If the **return_path_remove** configuration option is set (the default), Exim removes **Return-path:** headers from incoming messages.

43.11 The Sender: header

For locally-originated messages, unless originated by a trusted user using the **-f** or **-bs** command line option, any existing **Sender:** header is removed. For non-trusted callers, a check is made to see if the address given in the **From:** header is the correct (local) sender of the message. If not, a **Sender:** header giving the true sender address is added to the message. No processing of the **Sender:** header is done for messages originating externally.

43.12 The To: header

If a message has no **To:**, **Cc:**, or **Bcc:** header, Exim adds an empty **Bcc:** header, in accordance with RFC 822, except when the message is being received locally with the recipients supplied on the command line. In this case, a **To:** header listing the recipients is normally added. Some mailing list software is known to submit messages in this way, and in this case the creation of a **To:** header is not what is wanted. If the **always_bcc** option is set, Exim adds an empty **Bcc:** header instead in this circumstance. An **Apparently-to:** header is never added.

43.13 Adding and removing headers

The addition and removal of headers can be specified on any of the drivers, and also in system filter files. Changes specified in the system filter affect all deliveries of a message.

Header changes specified on a director or router affect all addresses handled by that driver, and also any new addresses it generates. If an address passes through several directors and/or routers, the changes are cumulative. When a message is processed by a transport, the message's original set of headers is output, except for those named in any **headers_remove** options that the address has encountered as it was processed, and any in the transport's own **headers_remove** option. Then any new headers from any **headers_add** options are output.

43.14 Constructed addresses

When Exim constructs a sender address for a locally-generated message, it uses the form

<*user name*> <<*login*>@<*qualify_domain*>>

For example:

```
Zaphod Beeblebrox <zaphod@end.univ>
```

The user name is obtained from the **-F** command line option if set, or otherwise by looking up the calling user by **getpwuid()** and extracing the 'gecos' field from the password entry. If the 'gecos' field contains an ampersand character, this is replaced by the login name with the first letter upper-cased, as is conventional in a number of operating systems. See the **gecos_name** option for a way to tailor the handling of the 'gecos' field. The **unknown_username** option can be used to specify user names in cases when there is no password file entry. In all cases the user name is made to conform to RFC 822 by quoting all or parts of it if necessary.

43.15 Case of local parts

RFC 822 states that the case of letters in the local parts of addresses cannot be assumed not to be significant. Exim preserves the case of local parts of remote addresses. However, when it is processing an address in one of its local domains, the case of letters in the local part is significant only when **locally_caseless** is unset. This option is set by default, and this causes Exim to lowercase local parts in local domains before processing them.

If you must have mixed-case user names in your password file, the best way to proceed, assuming you want case-independent handling of incoming email, is to unset **locally_caseless** and then set up an initial **smartuser** director to convert incoming local parts to the correct case by a file lookup such as

```
new_address = "${lookup{${lc:$local_part}}cdb\
               {/etc/usercased.cdb}{$value}fail}\
               @$domain"
```

43.16 Dots in local parts

RFC 822 forbids empty components in local parts. That is, an unquoted local part may not begin or end with a dot, nor have two consecutive dots in the middle. However, it seems that many MTAs do not enforce this, so Exim permits empty components for compatibility.

43.17 Rewriting addresses

Rewriting of sender and recipient addresses, and addresses in headers, can happen automatically, or as the result of configuration options, as described in chapter 32. The headers that may be affected by this are **Bcc:**, **Cc:**, **From:**, **Reply-to:**, **Sender:**, and **To:**.

Automatic rewriting includes qualification, as mentioned above. The other case in which it can happen is when an incomplete non-local domain is given. The routing process may cause this to be expanded into the full domain name. For example, a header such as

```
To: hare@teaparty
```

might get rewritten as

```
To: hare@teaparty.wonderland.fict.book
```

Rewriting as a result of routing is the one kind of message processing that does not happen at input time, as it cannot be done until the address has been routed.

Strictly, one should not do *any* deliveries of a message until all its addresses have been routed, in case any of the headers get changed as a result of routing. However, doing this in practice would hold up many deliveries for unreasonable amounts of time, just because one address could not immediately be routed. Exim therefore does not delay other deliveries when routing of one or more addresses is deferred.

44. Automatic mail processing

This chapter describes some of the ways in which incoming mail can be processed automatically, either on a system-wide basis, or as specified by individual users.

44.1 System-wide automatic processing

Simple re-addressing of messages can be handled by **aliasfile** or **forwardfile** directors. The particular case of mailing lists is covered in chapter 36. Other kinds of automatic processing can be handled by suitable configurations of directors and transports. As an example, here is an extract from the configuration of a system which tries to send back helpful information when a message is received for an unknown user. The last director in the configuration is:

```
unknownuser:
  driver = smartuser
  transport = unknownuser_pipe
  no_verify
```

This collects all the addresses whose local parts haven't been matched by any other director, and directs them to a special pipe transport, whose configuration is:

```
unknownuser_pipe:
  driver = pipe
  command = /opt/exim/util/baduser.sh
  ignore_status
  return_output
  user = nobody
```

The script is run as the user 'nobody', and it applies heuristics and a *soundex* search to the local part, in an attempt to produce a list of possible users for whom the message might have been intended. This is then included in a message that is written to its standard output; Exim picks this up and returns it to the sender as part of the delivery error message.

Chapter 41 describes how to arrange to run a system filter file once per message. Sometimes there is a requirement to set up similar automatic processing, but on a per-address basis, that is, the filter is run once for each address. This can be done by using a director such as the following:

```
filter_per_address:
  driver = forwardfile
  no_verify
  filter
  file = /etc/per-address-filter
  no_check_local_user
  user = nobody
```

See the separate document entitled *Exim's User interface to mail filtering* which describes the available filtering commands. Care should be taken to ensure that none of the commands in the filter file specify a significant delivery if the message is to go on to be delivered to its intended recipient. The director will not then claim to have dealt with the address, so it will be passed on to subsequent directors to be delivered in the normal way. Note that a traditional (non-filter) **.forward** file does not have this property, so cannot be used in this way, though you could use it to forward all mail for a particular domain to a single recipient in a different domain.

44.2 Taking copies of mail

Some installations have policies that require archive copies of all messages to be made. A single copy of each message can easily be taken by an appropriate command in a system filter, which could, for example, use a different file for each day's messages.

There is also a shadow transport mechanism that can be used to take copies of messages that are successfully delivered by local transports, one copy per delivery. This could be used, *inter alia*, to implement automatic notification of delivery by sites that insist on doing such things.

44.3 Automatic processing by users

Users can cause their mail to be processed automatically by creating **.forward** files, provided that Exim's configuration contains an appropriate **forwardfile** director. Traditionally, such files contain just a list of forwarding addresses, local files, and pipe commands, but if the **forwardfile** director has the **filter** option set, users can access Exim's filtering facilities by beginning a **.forward** file with the text '# Exim filter'. Details of the syntax and semantics of filter files are described in a separate document entitled *Exim's User interface to mail filtering*; this is intended for use by end users.

The name **.forward** is purely conventional; a **forwardfile** director can be configured to use any arbitrary name. As there are some finger daemons that display the contents of users' **.forward** files, some sites might like to use a different name when mail filtering is provided.

What users may do in their **.forward** files can be constrained by various options of the **forwardfile** director:

- If the **filter** option is not set, then only traditional **.forward** files are permitted.
- If the **forbid_file** option is set, then neither a traditional **.forward** file, nor a filter file may direct that a message be appended to a particular local file. An attempt to do so causes a delivery error.
- If the **forbid_filter_log** option is set, then the use of the **log** command in a filter file is not permitted.
- If the **forbid_pipe** option is set, then neither a traditional **.forward** file, nor a filter file may direct that a message be piped to a user-specified command. An attempt to do so causes a delivery error.
- If the **forbid_reply** option is set, then a filter file may not direct that a new mail message be created. An attempt to do so causes a delivery error.

If piping is permitted, the pipe transport that is used (conventionally called **address_pipe**) can constrain the command to be taken from a particular set of files. Pipe commands generated from traditional **.forward** files are not string-expanded, but when a pipe command is generated in a filter file, each argument is separately expanded.

If delivery to specified files is permitted, the **appendfile** transport that is used (conventionally called **address_file**) can specify that the file must already exist, or can restrict the whereabouts of its creation by means of the **create_file** option.

44.4 Simplified vacation processing

The traditional way of running the **vacation** program is for a user to set up a pipe command in a **.forward** file. This is prone to error by inexperienced users. There are two features of Exim that can be used to make this process simpler for users:

- A local part prefix such as 'vacation-' can be specified on a director which causes the message to be delivered directly to the **vacation** program, or uses Exim's **autoreply** transport. The contents of a user's **.forward** file are then much simpler. For example:

  ```
  spqr, vacation-spqr
  ```

- The **require_files** generic director option can be used to trigger a vacation delivery by checking for the existence of a certain file in the user's home directory. The **unseen** generic option should also be used, to ensure that the original delivery also proceeds. In this case, all the user has to do is to create a file called, say, **.vacation**, containing a vacation message.

Another advantage of both these methods is that they both work even when the use of arbitrary pipes by users is locked out.

45. Log files

Exim writes four different log files called **mainlog**, **rejectlog**, **paniclog**, and **processlog** into a sub-directory of its spool directory called **log**, unless a compile-time option called LOG_FILE_PATH or a runtime option called **log_file_path** is defined to specify a different directory and template for the file names. The template has a wild portion which is replaced by 'main', 'reject', 'panic', or 'process' when writing to a log file. See the comments in **src/EDITME** for more details of this. In the text below, the default file names are used.

- The file called **mainlog** records the arrival of each message and each delivery in a single line in each case. The format is as compact as possible, in an attempt to keep down the size of log files. Two-character flag sequences make it easy to pick out these lines. A number of other events are also recorded in the main log. Some of these entries can be suppressed by changing the value of the **log_level** configuration option.
- The file called **rejectlog** records information from messages that are rejected as a result of a configuration option (that is, for policy reasons) for example, because their return paths are invalid. In this particular case, the headers are written to this log, following a copy of the one-line message that is also written to the main log. For other rejections, just a single line is written to the reject log.
- The file called **paniclog** is written when Exim suffers a disaster and has to bomb out. This should be checked regularly to pick up any problems. When exim cannot open its panic log, it tries as a last resort to write to the system log. This is opened with LOG_PID+LOG_CONS and the facility code of LOG_MAIL. The message itself is written at priority LOG_CRIT.
- On systems that support signal handlers that restart a system call on exit, Exim reacts to a SIGUSR1 signal by writing a line describing its current activity to a file called **processlog**. This makes it possible to find out what each exim process on a machine is currently doing.

A utility script called **exicyclog** which renames and compresses the main and reject logs each time it is called is provided. The maximum number of old logs to keep can be set. It is suggested this is run as a daily **cron** job. A Perl script called **eximstats** which does simple analysis of main log files is also provided. See chapter 47 for details of both these utilities.

An Exim delivery process opens the main log when it first needs to write to it, and it keeps the file open in case subsequent entries are required – for example, if a number of different deliveries are being done for the same message. However, remote SMTP deliveries can take a long time, and this means that the file may be kept open long after it is renamed if **exicyclog** or something similar is being used to rename log files on a regular basis. To ensure that a switch of log files is noticed as soon as possible, Exim calls **stat()** on the main log's name before reusing an open file, and if the file does not exist, or its inode has changed, the old file is closed and Exim tries to open the main log from scratch. Thus, an old log file may remain open for quite some time, but no Exim processes should write to it once it has been renamed.

45.1 Logging message reception

The format of the single-line entry in the main log that is written for every message received is shown in the example below, which is split over several lines in order to fit it on the page:

```
1995-10-31 08:57:53 0tACW1-0005MB-00 <= kryten@dwarf.fict.book
  H=mailer.fict.book [123.123.123.123] U=exim
  P=smtp S=5678 id=<incoming message id>
```

The H and U fields identify the remote host and record the RFC 1413 identity of the user that sent the message, if one was received. The number given in square brackets is the IP address of the sending host. If there is just a single host name in the H field, as above, it has been verified to correspond to the IP address (see the **host_lookup** option). If the name is in parentheses, it was the name quoted by the remote host in the SMTP HELO or EHLO command, and has not been verified. If verification yields a different name to that given for HELO or EHLO, then the verified name appears first, followed by the HELO or EHLO name in parentheses.

Misconfigured hosts (and mail forgers) sometimes put an IP address, with or without brackets, in the HELO or EHLO command, leading to entries in the log containing things like

```
  H=(10.21.32.43) [123.99.8.34]
  H=([10.21.32.43]) [123.99.8.34]
```

which can be confusing. Only the final address in square brackets can be relied on. For locally generated messages, the H field is omitted, and the U field contains the login name of the caller of Exim.

For all messages, the P field specifies the protocol used to receive the message, and the id field records the existing message id, if present. The size of the received message is given by the S field. When the message is delivered, headers may get removed or added, so that the size of delivered copies of the message may not correspond with this value (and indeed may be different to each other).

If the **log_received_sender** option is on, the unrewritten original sender of a message is added to the end of the log line that records the message's arrival, after the word 'from'. If the **log_received_recipients** option is on, a list of all the recipients of a message is added to the log line, preceded by the word 'for'. This happens after any unqualified addresses are qualified, but before any rewriting is done. If the **log_subject** option is on, the subject of the message is added to the log line, preceded by 'T=' (T for 'topic', since S is already used for 'size').

A delivery error message is shown with the sender address '<>', and if it is a locally-generated error message, this is normally followed by an item of the form

  `R=`<*message id*>

which is a reference to the local identification of the message that caused the error message to be sent.

45.2 Logging deliveries

The format of the single-line entry in the main log that is written for every delivery is shown in one of the examples below, for local and remote deliveries, respectively. Each example has been split into two lines in order to fit it on the page:

```
1995-10-31 08:59:13 0tACW1-0005MB-00 => marv <marv@hitch.fict.book>
  D=localuser T=local_delivery
1995-10-31 09:00:10 0tACW1-0005MB-00 => monk@holistic.fict.book
  R=lookuphost T=smtp H=holistic.fict.book [234.234.234.234]
```

For ordinary local deliveries, the original address is given in angle brackets after the final delivery address, which might be a pipe or a file. If intermediate address(es) exist between the original and the final address, the last of these is given in parentheses after the final address. However, **log_all_parents** can be set to cause all intermediate addresses to be logged.

If a shadow transport was run after a successful local delivery, the log line for the successful delivery has an item added on the end, of the form

  `ST=`<*shadow transport name*>

If the shadow transport did not succeed, the error message is put in parentheses afterwards.

When a local delivery occurs as a result of routing rather than directing (for example, messages are being batched up for transmission by some other means), the log entry looks more like that for a remote delivery.

For normal remote deliveries, if the **log_smtp_confirmation** option is on, the response to the final '.' in the SMTP transmission is added to the log line, preceded by 'C='. If the final delivery address is not the same as the original address (owing to changes made by routers), the original is shown in angle brackets.

The generation of a reply message by a filter file gets logged as a ‘delivery’ to the addressee, preceded by ‘>’. The D and T items record the director and transport. For remote deliveries, the router, transport, and host are recorded.

When more than one address is included in a single delivery (for example, two SMTP RCPT commands in one transaction) then the second and subsequent addresses are flagged with ‘->’ instead of ‘=>’. When two or more messages are delivered down a single SMTP connection, an asterisk follows the IP address in the log lines for the second and subsequent messages.

When the **-N** debugging option is used to prevent delivery from actually occurring, log entries are flagged with ‘*>’ instead of ‘=>’.

When a message is discarded as a result of the command ‘seen finish’ being obeyed in a filter file which generates no deliveries, a log entry of the form

```
  1998-12-10 00:50:49 0znuJc-0001UB-00 => discarded
    <low.club@trick4.bridge> D=userforward
```

is written, to record why no deliveries are logged.

45.3 Deferred deliveries

When a delivery is deferred, a line of the following form is logged:

```
1995-12-19 16:20:23 0tRiQz-0002Q5-00 == marvin@endrest.book
  T=smtp defer (146): Connection refused
```

In the case of remote deliveries, the error is the one that was given for the last IP address that was tried. Details of individual SMTP failures are also written to the log, so the above line would be preceded by something like

```
1995-12-19 16:20:23 0tRiQz-0002Q5-00 Failed to connect to endrest.book
  [239.239.239.239]: Connection refused
```

When a deferred address is skipped because its retry time has not been reached, a message is written to the log, but this can be suppressed by changing the **log_level** option.

45.4 Delivery failures

If a delivery fails, a line of the following form is logged:

```
1995-12-19 16:20:23 0tRiQz-0002Q5-00 ** jim@trek99.film
  <jimtrek99.film>: unknown mail domain
```

This is followed (eventually) by a line giving the address to which the delivery error has been sent.

45.5 Completion

A line of the form

```
  1995-10-31 09:00:11 0tACW1-0005MB-00 Completed
```

is written to the main log when a message is about to be removed from the spool at the end of its processing.

45.6 Other log entries

Various other types of log entry are written from time to time. Most should be self-explanatory. Among the more common are:

- *retry time not reached* An address previously suffered a temporary error during directing or routing or local delivery, and the time to retry it has not yet arrived.
- *retry time not reached for any host* An address previously suffered temporary errors during remote delivery, and the retry time has not yet arrived for any of the hosts to which it is routed.

- *spool file locked* An attempt to deliver a message cannot proceed because some other Exim process is already working on the message. This can be quite common if queue running processes are started at frequent intervals. The **exiwhat** utility script can be used to find out what Exim processes are doing.

45.7 Log level

The **log_level** configuration option controls the amount of data written to the main log. The higher the number, the more is written. A value of 6 causes all possible messages to appear, though higher levels may get defined in the future. Zero sets a minimal level of logging, with higher levels adding the following, successively:

1 rejections because of policy
re-addressing by the system filter

2 rejections because of message size

3 verification failures

4 SMTP timeouts
SMTP connection refusals because too busy
SMTP unexpected connection loss
SMTP (dis)connections when **log_smtp_connections** is set
SMTP syntax errors when **log_smtp_syntax_errors** is set
non-immediate delivery of SMTP messages because of load level,
or number of connections etc.

5 'retry time not reached [for any host]'
'spool file locked' (i.e. some other process is delivering the message)
'message is frozen' (when skipping it in a queue run)
'error message sent to ...'

6 invalid HELO and EHLO arguments (see **helo_verify**)

The default log level is 5, which is on the verbose side. Rejection information is still written to the reject log in all cases.

45.8 Message log

In addition to the four main log files, Exim writes a log file for each message that it handles. The names of these per-message logs are the message ids, and they are kept in the **msglog** sub-directory of the spool directory. A single line is written to the message log for each delivery attempt for each address. It records either a successful delivery, or the reason (temporary or permanent) for failure. If the log level is 5 or higher, 'retry time not reached' messages are also written to individual message logs. If the log level is 4 or less, they are suppressed after the first delivery attempt.

When a local part is expanded by aliasing or a forwarding file, a line is written to the message log when all its child deliveries are completed. SMTP connection failures for each remote host are also logged here. The log is deleted when processing of the message is complete, unless **preserve_message_logs** is set, but this should be used only with great care because they can fill up your disc very quickly.

46. Day-to-day management

This chapter describes some of the regular tasks that need to be done to keep Exim running smoothly.

46.1 The panic log

When certain disasters occur, Exim writes entries to its panic log. These are often copied to the main log as well, but can get lost amid the mass of other entries. The panic log should be empty under normal circumstances. It is therefore a good idea to check it (or to have a **cron** script check it) regularly, in order to become aware of any problems.

46.2 The reject log

If checking of sender addresses on incoming mail is enabled, the headers of rejected messages are written to the reject log. Other policy rejections also cause entries in this log, which should be regularly inspected to ensure that the checking is working properly, and to pick up errors such as missing DNS entries.

46.3 Log cycling

The **exicyclog** script (see chapter 47) cycles the names of log files, compresses all but the most recent, and deletes the oldest. This should be run at intervals dependent on the amount of mail traffic. For a system with a reasonable amount of mail, running it daily via **cron** is suggested.

46.4 Statistics

The **eximstats** script (see chapter 47) produces statistics about messages received and delivered, by analysing log files.

46.5 What is Exim doing?

On systems that can restart a system call after receiving a signal, Exim responds to the SIGUSR1 signal by writing a line describing what it is doing to its process log. The **exiwhat** script (see chapter 47) sends the signal to all Exim processes it can find, having first emptied the process log. It then waits for one second to allow the Exim processes to react before displaying the results. In order to run **exiwhat** successfully you have to have sufficient privilege to send the signal to the Exim processes, so it is normally run as root.

When the number of processes handling incoming SMTP calls is limited by setting the **smtp_accept_max** option, the daemon uses the SIGCHLD signal to detect when any of its subprocesses finishes. On some operating systems this signal sometimes gets lost when the system is very busy. However, Exim's daemon cleans up subprocesses every time it wakes up, so even if SIGCHLD doesn't happen, the completion of subprocesses should eventually get noticed.

46.6 Changing the configuration

A changed configuration file is picked up immediately by any Exim processes that are subsequently started, and by any existing process that re-execs Exim, but it will not be noticed by any existing processes. The daemon process can be caused to restart itself by sending it the SIGHUP signal, which should also be sent when a new version of the Exim binary is installed. Restarting causes its process id to change. The current process id is written to a file whose name depends on the type of daemon being run. By default, the file is written in Exim's spool directory, but a compile-time configuration of PID_FILE_PATH can be used to cause it to be placed elsewhere. When the daemon is both listening for incoming SMTP on the standard port and periodically starting queue runner processes, the file is called **exim-daemon.pid**. If it is doing only one of these things, the option that started it (either **-bd** or **-q**<*time*>) is added to the file name. It is not necessary to use SIGHUP when changing the contents of any files referred to in the configuration (for example, alias files) since each delivery process reads such files independently.

46.7 Watching the queue

The queue of messages awaiting delivery can be examined by running the Exim monitor (see chapter 48), or by obeying **exim -bp** periodically. The **exiqsumm** utility script can be called to obtain a summary of the waiting messages for each domain, sorted by domain, age, or message count.

If any messages are frozen, their header files and message log files should be examined to determine the cause of the problem. Once the problem is believed to be fixed, the messages can be unfrozen by the administrator, who can also kick off an immediate delivery attempt, and also change recipient and sender addresses if necessary.

46.8 Holding domains

The option **hold_domains** allows mail for particular domains to be held on the queue manually. This option is intended as a temporary operational measure for delaying the delivery of mail while some problem is being sorted out, or some new configuration tested.

47. Exim utilities

A number of utility scripts and programs are supplied with Exim. Most of them are built as part of the normal building process, but the log file analyser is entirely free-standing.

47.1 Querying Exim processes

The shell script called **exiwhat** first of all empties the process log file in Exim's log directory. It then uses the **ps** command to find all processes running exim, and sends each one the SIGUSR1 signal. This causes each process to write a single line describing its current activity to the process log. The script then waits for one second to allow the Exim processes to react, then copies the file to the standard output.

Unfortunately, the **ps** command varies between different versions of Unix. Not only are different options used, but the format of the output is different. For this reason, there are some system configuration options that configure exactly how **exiwhat** works. If it doesn't seem to be working for you, check the following compile-time options:

`EXIWHAT_PS_CMD`	the command for running **ps**
`EXIWHAT_PS_ARG`	the argument for **ps**
`EXIWHAT_EGREP_ARG`	the argument for **egrep** to select from **ps** output
`EXIWHAT_KILL_ARG`	the argument for the **kill** command

This facility is available only in operating systems where a signal handler can be set up such that an interrupted system call is resumed when the signal handler has finished. An example of typical output from **exiwhat** is

```
  164 1.82 daemon: -q1h, listening on port 25
10483 1.90 running queue: waiting for 0tAycK-0002ij-00 (10492)
10492 1.90 delivering 0tAycK-0002ij-00 to mail.ref.book [42.42.42.42]
  (editor@ref.book)
10592 1.82 handling incoming call from [245.211.243.242]
10628 1.90 accepting a local non-SMTP message
```

The first number in the output line is the process number; the second is the Exim version number. The third line has been split here, in order to fit it on the page. Because Exim processes run under a variety of uids, it is necessary to run **exiwhat** as root in order to be able to send the signal to all Exim processes.

47.2 Summarising the queue

The **exiqsumm** utility is a Perl script, provided in the **util** directory, which reads the output of **exim -bp** and produces a summary of the messages by outputting a line like the following for each domain:

```
    3   2322   74m   66m  msn.com
```

This contains the number of messages for that domain, their total volume, and the length of time the oldest and the newest have been waiting. By default the output is sorted on the domain name, but **exiqsumm** has the options **-a** and **-c**, which cause it to be sorted by oldest message and by count of messages, respectively.

The output of **exim -bp** is based on the original addresses in the message, so no addresses generated by aliasing or forwarding are included. Consequently this applies also to the output from **exiqsumm**.

47.3 Extracting log information

The **exigrep** utility is a Perl script, provided in the **util** directory, that extracts from one or more log files all entries relevant to any message whose log entries contain at least one that matches a given pattern. The pattern match is case-insensitive. Thus one can search for all mail for a given user or a given host, for example. The usage is:

```
exigrep [-l] <pattern> [<log file>] ...
```

where the **-l** flag means 'literal', that is, treat all characters in the pattern as standing for themselves. Otherwise the pattern must be a Perl regular expression. If no file names are given on the command line, the standard input is read.

47.4 Cycling log files

The **exicyclog** script cycles the main and reject log files. Each time it is run the files get 'shuffled down' by one. If the main log file name is **mainlog** (the default) then when **exicyclog** is run **mainlog** becomes **mainlog.01**, the previous **mainlog.01** becomes **mainlog.02** and so on, up to a limit which is set in the script, and which defaults to 10.

In versions of Exim prior to 1.90, **exicyclog** used single-digits for numbers less than ten. This was changed to make the files list in a more natural order. The script contains conversion code. If it finds a file called **mainlog.1** it attempts to rename all files in the old form to the new form.

If no **mainlog** file exists, the script does nothing. Reject logs are handled similarly. Files that 'drop off' the end are deleted. All files with numbers greater than 01 are compressed, using a compression command which is configured in the script.

It is usual to run **exicyclog** daily from a **crontab** entry of the form

```
1 0 * * *  /opt/exim/bin/exicyclog
```

In this way, each day's log is (mostly) in a separate file. There will be some overlap from processes that have the log open at the time of renaming.

The **exicyclog** script can be run as the Exim user when one is defined, as the log files will be owned by that user in that case. Otherwise it has to be run as root.

47.5 Making DBM files

The **exim_dbmbuild** program reads an input file in the format of an alias file (see chapter 21) and writes a DBM database using the lower-cased alias names as keys and the remainder of the information as data. A terminating zero is included as part of the key string. The lower-casing can be prevented by calling it with the **-nolc** option. Two arguments are required: the name of the input file (which can be a single hyphen to indicate the standard input), and the base name of the output database. For example:

```
exim_dbmbuild /etc/aliases /etc/aliases
```

reads the system alias file and creates a DBM database using **/etc/aliases** as the base name. In systems that use the **ndbm** routines, the database consists of two files called (in this case) **/etc/aliases.dir** and **/etc/aliases.pag**, while in systems using the **ndbm** interface to the **db** routines a single file called **/etc/aliases.db** is created. If the native **db** interface is in use (USE_DB is set in a compile-time configuration file) then a single file with no added prefix is created. In this case the two file names on the command line must be different. The utility in fact creates the database under a temporary name, and then renames the file(s).

47.6 Individual retry times

A utility called **exinext** (mostly a Perl script) provides the ability to fish specific information out of the retry database. Given a mail domain (or a complete address), it looks up the hosts for that domain, and outputs any retry information. At present, the retry information is obtained by running **exim_dumpdb** (see below) and post-processing the output. For example:

```
exinext piglet@milne.fict.book
kanga.milne.fict.book:100.100.8.1 error 146: Connection refused
  first failed: 21-Feb-1996 14:57:34
  last tried:   21-Feb-1996 14:57:34
  next try at:  21-Feb-1996 15:02:34
roo.milne.fict.book:100.100.8.3 error 146: Connection refused
  first failed: 20-Jan-1996 13:12:08
  last tried:   21-Feb-1996 11:42:03
  next try at:  21-Feb-1996 19:42:03
  past final cutoff time
```

You can also give **exinext** a local **local_part**, without a domain, and it will give any retry information for it. Also, a message id can be given to obtain retry information pertaining to a specific message. This exists only when an attempt to deliver a message to a remote host suffers a message-specific error (see section 42.2). **Exinext** is not particularly ef ficient, but then it isn’ t expected to be run very often.

47.7 Database maintenance

Three utility programs are provided for maintaining the DBM files that Exim uses to contain its delivery hint information. Each program requires two arguments. The first specifies the name of Exim’s spool directory, and the second is the name of the database it is to operate on. These are as follows:

- **retry**: the database of retry information
- **reject**: the database of information about rejected messages
- **wait-**<*transport name*>: databases of information about messages waiting for remote hosts
- **serialize-**<*transport name*>: databases of information about current connections to hosts which are restricted to one connection at a time
- **serialize-etrn-runs**: database of information about current queue runs started by the ETRN command when **smtp_etrn_serialize** is set.

The entire contents of a database are written to the standard output by the **exim_dumpdb** program, which has no options or arguments other than the spool and database names. For example, to dump the retry database:

```
exim_dumpdb /var/spool/exim retry
```

Two lines of output are produced for each entry:

```
  T:mail.ref.book:242.242.242.242 146 77 Connection refused
31-Oct-1995 12:00:12  02-Nov-1995 12:21:39  02-Nov-1995 20:21:39 *
```

The first item on the first line is the key of the record. It starts with one of the letters D, R, or T, depending on whether it refers to a directing, routing, or transport retry. For a local delivery, the next part is the local address; for a remote delivery it is the name of the remote host, followed by its failing IP address. Then there follows an error code, an additional error code, and a textual description of the error.

The three times on the second line are the time of first failure, the time of the last delivery attempt, and the computed time for the next attempt. The line ends with an asterisk if the cutoff time for the last retry rule has been exceeded.

Each output line from **exim_dumpdb** for the reject database consists of a date and time, followed by the letter T or F and a fixed point number, followed by the address that was rejected, followed either by the name of the host that sent the bad address, if this has been verified, or otherwise the IP address. The letter is F if only one previous rejection of this address (from this host) has been done recently, and T if a second has occurred, causing rejection of the MAIL command, and subsequently rejection of the RCPT commands. The fixed point number is zero when the last rejection was a permanent one.

Otherwise it records the rate of temporary rejections for the same address from the same host, per hour.

Each output line from **exim_dumpdb** for the **wait**-*xxx* databases consists of a host name followed by a list of ids for messages that are or were waiting to be delivered to that host. If there are a very large number for any one host, continuation records, with a sequence number added to the host name, may be seen. The data in these records is often out of date, because a message may be routed to several alternative hosts, and Exim makes no effort to keep cross-references.

Each output line from **exim_dumpdb** for the **serialize-smtp** database consists of a host name preceded by the time that Exim made a connection to that host. Exim keeps track of connections only for those hosts or networks that have been configured for serialization.

The **exim_tidydb** utility program is used to tidy up the contents of the databases. If run with no options, it removes all records from a database that are more than 30 days old. The cutoff date can be altered by means of the **-t** option, which must be followed by a time. For example, to remove all records older than a week from the retry database:

```
  exim_tidydb -t 7d /var/spool/exim retry
```

For the **wait**-*xxx* databases, the **-f** option can also be used. This causes a check to be made to ensure that message ids in database records are those of messages that are still on the queue. Other message ids are removed, and if this leaves records empty, they are also removed. The **-f** option can also be used for the retry database; it causes the removal of any retry records for specific messages if those messages no longer exist. For other types of database, **-f** has no effect.

The **exim_tidydb** utility outputs comments on the standard output whenever it removes information from the database. It is suggested that it be run periodically on all three databases, but at a quiet time of day, since it requires a database to be locked (and therefore inaccessible to Exim) while it does its work.

The **exim_fixdb** program is a utility for interactively modifying databases. Its main use is for testing Exim, but it might also be occasionally useful for getting round problems in a live system. It has no options, and its interface is somewhat crude. On entry, it prompts for input with a right angle-bracket. A key of a database record can then be entered, and the data for that record is displayed.

If ‘d’ is typed at the next prompt, the entire record is deleted. For all except the **retry** database, that is the only operation that can be carried out. For the **retry** database, each field is output preceded by a number, and data for individual fields can be changed by typing the field number followed by new data, for example:

```
  > 4 951102:1000
```

resets the time of the next delivery attempt. Time values are given as a sequence of digit pairs for year, month, day, hour, and minute. Colons can be used as optional separators.

47.8 Mail statistics

A Perl script called **eximstats** is supplied in the **util** directory. This has been hacked about quite a bit over time. It now gives quite a lot of information by default, but there are options for suppressing various parts of it. Following any options, the arguments to the script are a list of files, which should be main log files.

Eximstats extracts information about the number and volume of messages received from or delivered to various hosts. The information is sorted both by message count and by volume, and the top fifty hosts in each category are listed on the standard output. For messages delivered and received locally, similar statistics are produced per user.

The output also includes total counts and statistics about delivery errors, and histograms showing the number of messages received and deliveries made in each hour of the day. A delivery with more than one address in its ‘envelope’ (for example, an SMTP transaction with more than one RCPT command) is counted as a single delivery by **eximstats**.

Though normally more deliveries than receipts are reported (as messages may have multiple recipients), it is possible for **eximstats** to report more messages received than delivered, even though the spool is empty at the start and end of the period in question. If an incoming message contains no valid recipients, no deliveries are recorded for it. An error report is handled as an entirely separate message.

Eximstats always outputs a grand total summary giving the volume and number of messages received and deliveries made, and the number of hosts involved in each case. It also outputs the number of messages that were delayed (that is, not completely delivered at the first attempt), and the number that had at least one address that failed.

The remainder of the output is in sections that can be independently disabled or modified by various options. It consists of a summary of deliveries by transport, histograms of messages received and delivered per time interval (default per hour), information about the time messages spent on the queue, a list of relayed messages, lists of the top fifty sending hosts, local senders, destination hosts, and destination local users by count and by volume, and a list of delivery errors that occurred.

The relay information lists messages that were actually relayed, that is, they came from a remote host and were directly delivered to some other remote host. A delivery that is considered as a relay by the checking features described in section 40.4, because its domain is not in **local_domains**, might still end up being delivered locally under some configurations, and if this happens it doesn't show up as a relay in the **eximstats** output.

The options for **eximstats** are as follows:

-nt Suppress the statistics about delivery by transport.

-h<*n*> This option controls the histograms of messages received and deliveries per time interval. By default the time interval is one hour. If **-h0** is given, the histograms are suppressed; otherwise the value of <*n*> gives the number of divisions per hour, so **-h2** sets an interval of 30 minutes, and the default is equivalent to **-h1**.

-q0 Suppress information about times messages spend on the queue.

-q<*n1*>**...**

This option sets an alternative list of time intervals for the queueing information. The values are separated by commas and are in seconds, but can involve arithmetic multipliers, so for example you can set 3∗60 to specify 3 minutes. A setting such as

```
-q60,5*60,10*60
```

causes **eximstats** to give counts of messages that stayed on the queue for less than one minute, less than five minutes, less than ten minutes, and over ten minutes.

-nr Suppress information about messages relayed through this host.

-nr/pattern/

Suppress information about relayed messages that match the pattern, which is matched against a string of the following form (split over two lines here in order to fit it on the page):

```
H=<host> [<ip address>] A=<sender address> =>
  H=<host> A=<recipient address>
```

for example

```
H=in.host [1.2.3.4] A=from@some.where =>
  H=out.host A=to@else.where
```

The sending host name appears in parentheses if it has not been verified as matching the IP address. The mail addresses are taken from the envelope, not the headers. This option allows you to screen out hosts whom you are happy to have using your host as a relay.

-t<*n*> Sets the 'top' count to <*n*>. This controls the listings of the 'top <*n*>' hosts and users by count and volume. The default is 50, and setting 0 suppresses the output altogether.

-tnl Omit local information from the 'top' listings.

-ne Suppress the list of delivery errors.

47.9 Mailbox maintenance

The **exim_lock** utility locks a mailbox file using the same algorithm as Exim. This can be used to prevent any modification of a mailbox by Exim or a user agent while investigating a problem. The utility requires the name of the file as its first argument. If the locking is successful, the second argument is run as a command (using C's **system()** function); if there is no second argument, the value of the SHELL environment variable is used; if this is unset or empty, **/bin/sh** is run. When the command finishes, the mailbox is unlocked and the utility ends. The following options are available:

-fcntl Use **fcntl()** locking on the open mailbox.

-lockfile Create a lock file before opening the mailbox.

-mbx Lock the mailbox using MBX rules.

-v Generate verbose output.

-q Suppress verification output.

If none of **-fcntl**, **-lockfile** or **-mbx** are given, the default is to create a lock file and also use **fcntl()** locking on the mailbox, which is the same as Exim's default. The use of **-fcntl** requires that the file be writeable; the use of **-lockfile** requires that the directory containing the file be writeable. Locking by lock file does not last for ever; Exim assumes that a lock file is expired if it is more than 30 minutes old.

The **-mbx** option is mutually exclusive with **-fcntl**. It causes a shared lock to be taken out on the open mailbox, and an exclusive lock on the file **/tmp/.***n***.***m* where *n* and *m* are the device number and inode number of the mailbox file. When the locking is released, if an exclusive lock can be obtained for the mailbox, the file in **/tmp** is deleted.

The default output contains verification of the locking that takes place. The **-v** option causes some additional information to be given. The **-q** option suppresses all output except error messages.

A command such as

```
exim_lock /var/spool/mail/spqr
```

runs an interactive shell while the file is locked, whereas

```
exim_lock -q /var/spool/mail/spqr <<End
<some commands>
End
```

runs a specific non-interactive sequence of commands while the file is locked, suppressing all verification output. A single command can be run by a command such as

```
exim_lock -q /var/spool/mail/spqr \
  "cp /var/spool/mail/spqr /some/where"
```

Note that if a command is supplied, it must be entirely contained within the second argument – hence the quotes.

48. The Exim monitor

The Exim monitor is an application which displays in an X window information about the state of Exim’s queue and what Exim is doing. An admin user can perform certain operations on messages from this GUI interface; however all such facilities are also available from the command line, and indeed, the monitor itself makes use of it.

48.1 Running the monitor

The monitor is started by running the script called **eximon**. This is a shell script which sets up a number of environment variables, and then runs the binary called **eximon.bin**. The appearance of the monitor window can be changed by editing the **Local/eximon.conf** file created by editing **exim_monitor/EDITME**. Comments in that file describe what the various parameters are for.

The parameters that get built into the **eximon** script can be overridden for a particular invocation by setting up environment variables of the same names, preceded by ‘`EXIMON_`’. For example, a shell command such as

```
EXIMON_LOG_DEPTH=400 eximon
```

(in a Bourne-compatible shell) runs **eximon** with an overriding setting of the LOG_DEPTH parameter. X resources can be used to change the appearance of the window in the normal way. For example, a resource setting of the form

```
Eximon*background: gray94
```

changes the colour of the background to light grey rather than white. The stripcharts are drawn with both the data lines and the reference lines in black. This means that the reference lines are not visible when on top of the data. However, their colour can be changed by setting a resource called ‘highlight’ (an odd name, but that’s what the Athena stripchart widget uses). For example, if your X server is running Unix, you could set up lighter reference lines in the stripcharts by obeying

```
xrdb -merge <<End
Eximon*highlight: gray
End
```

In order to see the contents of messages on the spool, and to operate on them, **eximon** must either be run as root or by an admin user, that is, a user who is a member of the Exim group (when one is defined).

The monitor’s window is divided into three parts. The first contains one or more stripcharts and two action buttons, the second contains a ‘tail’ of the main log file, and the third is a display of the queue of messages awaiting delivery.

48.2 The stripcharts

The first stripchart is always a count of messages on the queue. Its name can be configured by setting QUEUE_STRIPCHART_NAME in the **Local/eximon.conf** file. The remaining stripcharts are defined in the configuration script by regular expression matches on log file entries, making it possible to display, for example, counts of messages delivered to certain hosts or using certain transports. The supplied defaults display counts of received and delivered messages, and of local and SMTP deliveries. The default period between stripchart updates is one minute; this can be adjusted by a parameter in the **Local/eximon.conf** file.

The stripchart displays rescale themselves automatically as the value they are displaying changes. There are always 10 horizontal lines in each chart; the title string indicates the value of each division when it is greater than one. For example, ‘x2’ means that each division represents a value of 2.

It is also possible to have a stripchart which shows the percentage fullness of a particular disc partition, which is useful when local deliveries are confined to a single partition. This relies on the availability of the **statvfs** function or equivalent in the operating system. Most, but not all versions of Unix that support Exim have this. For this particular stripchart, the top of the chart always represents 100%, and the scale is given as ‘x10%’. It is configured by setting SIZE_STRIPCHART and (optionally) SIZE_STRIPCHART_NAME in the **Local/eximon.conf** file.

48.3 Main action buttons

Below the stripcharts there is an action button for quitting the monitor. Next to this is another button marked ‘Size’. They are placed here so that shrinking the window to its default minimum size leaves just the queue count stripchart and these two buttons visible. Pressing the ‘Size’ button causes the window to expand to its maximum size, unless it is already at the maximum, in which case it is reduced to its minimum.

When expanding to the maximum, if the window cannot be fully seen where it currently is, it is moved back to where it was the last time it was at full size. When it is expanding from its minimum size, the old position is remembered, and next time it is reduced to the minimum it is moved back there.

The idea is that you can keep a reduced window just showing one or two stripcharts at a convenient place on your screen, easily expand it to show the full window when required, and just as easily put it back to what it was. The idea is copied from what the **twm** window manager does for its **f.fullzoom** action. The minimum size of the window can be changed by setting the MIN_HEIGHT and MIN_WIDTH values in **Local/eximon.conf**.

Normally, the monitor starts up with the window at its full size, but it can be built so that it starts up with the window at its smallest size, by setting START_SMALL=yes in **Local/eximon.conf**.

48.4 The log display

The second section of the window is an area in which a display of the tail of the main log is maintained. This has a scroll bar at its lefthand side which can be used to move back to look at earlier text, and the arrow keys also have this effect. Similarly, there is a horizontal scroll bar for accessing long log lines. Text can be cut from this part of the window using the mouse in the normal way. The size of this subwindow is controlled by parameters in the configuration file **Local/eximon.conf**.

Searches of the text in the log window can be carried out by means of the ^R and ^S keystrokes, which default to a reverse and forwards search respectively. The search covers only the text that is displayed in the window. It cannot go further back up the log.

The point from which the search starts is indicated by a caret marker. This is normally at the end of the text in the window, but can be positioned explicitly by pointing and clicking with the left mouse button, and is moved automatically by a successful search. If new text arrives in the window when it is scrolled back, the caret remains where it is, but if the window is not scrolled back, the caret is moved to the end of the new text.

Pressing ^R or ^S pops up a window into which the search text can be typed. There are buttons for selecting forward or reverse searching, for carrying out the search, and for cancelling. If the ‘Search’ button is pressed, the search happens and the window remains so that further searches can be done. If the ‘Return’ key is pressed, a single search is done and the window is closed. If ^C is pressed the search is cancelled.

The searching facility is implemented using the facilities of the Athena text widget. By default this pops up a window containing both ‘search’ and ‘replace’ options. In order to suppress the unwanted ‘replace’ portion for eximon, a modified version of the **TextPop** widget is distributed with Exim. However, the linkers in BSDI and HP-UX seem unable to handle an externally provided version of **TextPop** when the remaining parts of the text widget come from the standard libraries. The compile-time option EXIMON_TEXTPOP can be unset to cut out the modified **TextPop**, making it possible to build Eximon on these systems, at the expense of having unwanted items in the search popup window.

48.5 The queue display

The bottom section of the monitor window contains a list of all messages that are on the queue, which includes those currently being received or delivered, as well as those awaiting delivery. The size of this subwindow is controlled by parameters in the configuration file **Local/eximon.conf**, and the frequency at which it is updated is controlled by another parameter in the same file – the default is 5 minutes, since queue scans can be quite expensive. However, there is an ‘Update’ action button just above the display which can be used to force an update of the queue display at any time.

When a host is down for some time, a lot of pending mail can build up for it, and this can make it hard to deal with other messages on the queue. To help with this situation there is a button next to ‘Update’ called ‘Hide’. If pressed, a dialogue box called ‘Hide addresses ending with’ is put up. If you type anything in here and press ‘Return’, the text is added to a chain of such texts, and if every undelivered address in a message matches at least one of the texts, the message is not displayed.

If there is an address that does not match any of the texts, all the addresses are displayed as normal. The matching happens on the ends of addresses so, for example, **cam.ac.uk** specifies all addresses in Cambridge, while **xxx@foo.com** specifies just one specific address. When any hiding has been set up, a button called ‘Unhide’ is displayed. If pressed, it cancels all hiding. Also, to ensure that hidden messages don’t get forgotten, a hide request is automatically cancelled after one hour.

While the dialogue box is displayed, you can’t press any buttons or do anything else to the monitor window. For this reason, if you want to cut text from the queue display to use in the dialogue box, you have to do the cutting before pressing the ‘Hide’ button.

The queue display contains, for each unhidden queued message, the length of time it has been on the queue, the size of the message, the message id, the message sender, and the first undelivered recipient, all on one line. If it is a delivery error message, the sender is shown as ‘<>’. If there is more than one recipient to which the message has not yet been delivered, subsequent ones are listed on additional lines, up to a maximum configured number, following which an ellipsis is displayed. Recipients that have already received the message are not shown. If a message is frozen, an asterisk is displayed at the left-hand side.

The queue display has a vertical scroll bar, and can also be scrolled by means of the arrow keys. Text can be cut from it using the mouse in the normal way. The text searching facilities, as described above for the log window, are also available, but the caret is always moved to the end of the text when the queue display is updated.

48.6 The queue menu

If the **shift** key is held down and the left button is clicked when the mouse pointer is over the text for any message, an action menu pops up, and the first line of the queue display for the message is highlighted. This does not affect any selected text. If you want to use some other event for popping up the menu, you can set the MENU_EVENT parameter in **Local/eximon.conf** to change the default, or set EXIMON_MENU_EVENT in the environment before starting the monitor. The value set in this parameter is a standard X event description. For example, to run eximon using **ctrl** rather than **shift** you could use

```
  EXIMON_MENU_EVENT='Ctrl<Btn1Down>' eximon
```

The title of the menu is the message id, and it contains entries which act as follows:

- **message log**: The contents of the message log for the message are displayed in a new text window.
- **headers**: Information from the spool file that contains the envelope information and headers is displayed in a new text window. See chapter 50 for a description of the format of spool files.
- **body**: The contents of the spool file containing the body of the message are displayed in a new text window. There is a default limit of 20,000 bytes to the amount of data displayed. This can be changed by setting the BODY_MAX option at compile time, or the EXIMON_BODY_MAX option at runtime.

- **deliver message**: A call to Exim is made using the **-M** option to request delivery of the message. This causes an automatic thaw if the message is frozen. The **-v** option is also set, and the output from Exim is displayed in a new text window. The delivery is run in a separate process, to avoid holding up the monitor while the delivery proceeds.
- **freeze message**: A call to Exim is made using the **-Mf** option to request that the message be frozen.
- **thaw message**: A call to Exim is made using the **-Mt** option to request that the message be thawed.
- **give up on msg**: A call to Exim is made using the **-Mg** option to request that Exim gives up trying to deliver the message. A delivery failure report is generated for any remaining undelivered addresses.
- **remove message**: A call to Exim is made using the **-Mrm** option to request that the message be deleted from the system without generating any failure reports.
- **add recipient**: A dialog box is displayed into which a recipient address can be typed. If the address is not qualified and the QUALIFY_DOMAIN parameter is set in **Local/eximon.conf**, the address is qualified with that domain. Otherwise it must be entered as a fully qualified address. Pressing RETURN causes a call to Exim to be made using the **-Mar** option to request that an additional recipient be added to the message, unless the entry box is empty, in which case no action is taken.
- **mark delivered**: A dialog box is displayed into which a recipient address can be typed. If the address is not qualified and the QUALIFY_DOMAIN parameter is set in **Local/eximon.conf**, the address is qualified with that domain. Otherwise it must be entered as a fully qualified address. Pressing RETURN causes a call to Exim to be made using the **-Mmd** option to mark the given recipient address as already delivered, unless the entry box is empty, in which case no action is taken.
- **mark all delivered**: A call to Exim is made using the **-Mmad** option to mark all recipient addresses as already delivered.
- **edit sender**: A dialog box is displayed initialized with the current sender's address. Pressing RETURN causes a call to Exim to be made using the **-Mes** option to replace the sender address, unless the entry box is empty, in which case no action is taken. If the address is not qualified and the QUALIFY_DOMAIN parameter is set in **Local/eximon.conf**, the address is qualified with that domain. Otherwise it must be a fully qualified address.
- **edit body**: A new xterm process is forked in which a call to Exim is made using the **-Meb** option in order to allow the body of the message to be edited. Note that the first line of the body file is the name of the file, and this should never be changed.

In cases when a call to Exim is made, the actual command used is reflected in a new text window by default, but this can be turned off for all except the delivery action by setting ACTION_OUTPUT=no in **Local/eximon.conf**. However, if the call results in any output from Exim (in particular, if the command fails) a window containing the command and the output is displayed. Otherwise, the results of the action are normally apparent from the log and queue displays. The latter is automatically updated for actions such as freezing and thawing, unless ACTION_QUEUE_UPDATE=no has been set in **Local/eximon.conf**. In this case the 'Update' button has to be used to force an update to the display after freezing or thawing.

In any text window that is displayed as result of a menu action, the normal cut-and-paste facility is available, and searching can be carried out using ^R and ^S, as described above for the log tail window.

49. Security considerations

This chapter discusses a number of issues concerned with security, some of which are also covered in other parts of this manual.

For reasons that this author does not understand, some people have promoted Exim as a ‘particularly secure’ mailer. Perhaps it is because of the existence of this chapter in the documentation. However, the intent of the chapter is simply to describe the way Exim works in relation to certain security concerns, not to make any specific claims about the effectiveness of its security as compared with other MTAs.

What follows is a description of the way Exim is supposed to be. Best efforts have been made to try to ensure that the code agrees with the theory, but an absence of bugs can never be guaranteed. Any that are reported will get fixed as soon as possible.

49.1 Root privilege

The Exim binary is normally setuid to root. In some special cases (for example, when the daemon is not in use and there are no conventional local deliveries), it may be possible to run it setuid to some user other than root. However, root privilege is usually required for two things:

- To set up a socket connected to the standard SMTP port (25) when initialising the listening daemon.
- To be able to change uid and gid in order to read forward files and perform local deliveries as the receiving user or as specified in the configuration.

It is not necessary to be root to do any of the other things Exim does, such as receiving messages and delivering them externally over SMTP, and it is obviously more secure if Exim does not run as root except when necessary.

If no user is specified for Exim in either the compile-time or runtime configuration files, then it runs as root all the time, except when performing local deliveries. When an alternative user is specified (which is recommended), it gives up root privilege when it can. Exactly how and when it does this depends on whether the operating system supports the **seteuid()** or the **setresuid()** function.

To avoid unnecessary complication, the discussion below talks about users, and functions for setting the uid. It should be understood that in all cases there is a corresponding group and gid, and that this is also changed whenever the uid is changed. The description is written in terms of **seteuid()**, since this is more common than **setresuid()**. However, it is possible to specify at compile time that an operating system has **setresuid()** and not **seteuid()**.

On systems without **seteuid()**, Exim uses **setuid()** to give up root privilege at certain times, at the expense of having to re-invoke itself (using **exec**) in order to regain privilege when necessary. If **seteuid()** is available, there is a configuration choice as to which method is used for temporarily giving up the privilege. Using **setuid()** is more secure, and is the default, but uses more resources.

There are two instances in which Exim always uses **setuid()**:

- Exim always uses **setuid()** to become a non-root user when running a local delivery process. There are no exceptions. This applies whether or not an Exim user is defined.
- Exim always uses **setuid()** to change to the Exim user (if one is defined) before doing remote deliveries. These are the last things a delivery process does, so it does not need to regain root privilege again.

There are two instances in which Exim always uses **seteuid()** (provided it is available in the operating system):

- When reading a user’s **.forward** file, Exim uses **seteuid()** to become that user. This is necessary when the file is not publicly readable and is on a remote NFS file system that is mounted without root privilege. If this is the case on a system without **seteuid()**, the **.forward** file cannot be read.
- If any director or router has the **require_files** option set to check the existence of a file as a specific user, then **seteuid()** is used to become that user for the duration of the check.

For other operations, the **security** configuration option controls whether Exim uses **setuid()** or **seteuid()** to change to its own uid. It can be set to one of three strings:

- **seteuid**: Exim uses **seteuid()** to give up root temporarily when it does not need it, and to regain the privilege subsequently. This enables it to run with a non-root effective uid most of the time, at very little cost, but offers less security.
- **setuid**: Exim uses **setuid()** to give up root when it is receiving a locally generated message, and after it has set up a listening socket when running as a daemon. This means that, in order to deliver a message that it has received, it has to re-invoke a fresh copy of itself to regain root privilege. During delivery, it retains root except when actually transporting the message. In particular, it runs the directors and routers as root. **Setuid()** is generally reckoned to be more secure than **seteuid()** but running this way uses more resources.
- **setuid+seteuid**: Exim uses **setuid()** as described immediately above, but in addition, it uses **seteuid()** to give up root privilege temporarily when it needs to regain it subsequently without losing a lot of state information, for example, while running the directors and routers.

On systems that do not support the **seteuid()** function, the only possible value for the **security** option is ‘setuid’, and this is the default on such systems if an Exim user is defined. Otherwise the default is ‘setuid+seteuid’ – the most secure setting.

49.2 Reading forward files

When forward files are read from users’ home directories and those home directories are NFS mounted without root privilege, even a program running as root cannot read a forward file that does not have world read access.

If the **seteuid()** function is being used as described in the previous section, so that Exim is not root when running the directors, then the **forwardfile** director automatically uses **seteuid()** to become the local user when attempting to read a **.forward** file in a user’s home directory. If **seteuid()** is not being used generally, but is available in the operating system, the **forwardfile** director can be configured to make use of it when reading files in home directories.

The **forwardfile** director does not necessarily have to read from users’ home directories as obtained from **getpwnam()**. It can be given a directory explicitly, and a specific associated user and group. The above remarks are applicable in this case also.

On systems that do not have **seteuid()**, the only way to support forward files on NFS file systems that do not export root is to insist that the files be world readable.

Forward files are permitted to contain :include: items unless forbidden by setting **forbid_include** in the director. If **seteuid()** is being used to read the forward file, then any included files are read as the same user. Otherwise Exim is running as root, and it insists that any included files are within the same directory as the forward file, and that there are no symbolic links below the directory. If no directory is specified (either explicitly or by looking up a local user’s home directory) then included files are not permitted when **seteuid()** is not in use.

When the filtering option is enabled for forward files, users can construct pipe commands that contain data from the incoming message by quoting variables such as **$sender_address**. To prevent the contents of inserted data from interfering with a command, the string expansion is done after the command line is split up into separate arguments, and the command is run directly instead of passing the command line to a shell.

49.3 Delivering to local files

Full details of the checks applied by **appendfile** before it writes to a file are given in chapter 15.

49.4 IPv4 source routing

Many operating systems suppress IP source-routed packets in the kernel, but some cannot be made to do this. Exim is configured by default to log incoming IPv4 source-routed TCP calls, and then to drop the call. These actions can be independently turned off. Alternatively, the IP options can be deleted instead of dropping the call. Things are all different in IPv6. No special checking is currently done.

49.5 The VRFY, EXPN, and ETRN commands in SMTP

Support for these SMTP commands is disabled by default. The VRFY command can be enabled by setting **smtp_verify**. The EXPN command can be enabled for specific hosts by setting **smtp_expn_hosts**, and there is a similar option controlling ETRN.

49.6 Privileged users

Exim recognises two sets of users with special privileges. Trusted users are able to submit new messages to Exim locally, but supply their own sender addresses and information about a sending host. For other users submitting local messages, Exim sets up the sender address from the uid, and doesn't permit a remote host to be specified.

However, an untrusted user is permitted to use the **-f** command line option in the special form **-f** <> to indicate that a delivery failure for the message should not cause an error report. This affects the message's envelope, but it does not affect the **Sender:** header.

Trusted users are used to run processes that receive mail messages from some other mail domain and pass them on to Exim for delivery either locally, or over the Internet. Exim trusts a caller that is running as root, as the Exim user (if defined), as any user listed in the **trusted_users** configuration option, or under any group listed in the **trusted_groups** option.

Admin users are permitted to do things to the messages on Exim's queue. They can freeze or thaw messages, cause them to be returned to their senders, remove them entirely, or modify them in various ways. In addition, admin users can run the Exim monitor and see all the information it is capable of providing, which includes the contents of files on the spool.

By default, the use of the **-M** and **-q** options to cause Exim to attempt delivery of messages on its queue is restricted to admin users. However, this restriction can be relaxed by setting the **no_prod_requires_admin** option.

Exim recognises an admin user if the calling process is running as root or as the Exim user (if defined) or if any of the groups associated with the calling process is the Exim group (if defined). It is not necessary actually to be running under the Exim group. However, if admin users who are not root or exim are to access the contents of files on the spool via the Exim monitor (which runs unprivileged), Exim must be built to allow group read access to its spool files.

49.7 Spool files

If a uid and gid are defined for Exim, then the spool directory and everything it contains will be owned by exim and have its group set to exim. The mode for spool files is defined in the **Local/Makefile** configuration file, and defaults to 0600. This should normally be changed to 0640 if a uid and gid are defined for Exim, to allow access to spool files via the Exim monitor by other members of the exim group.

49.8 Use of argv[0]

Exim examines the last component of **argv[0]**, and if it matches one of a set of specific strings, Exim assumes certain options. For example, calling Exim with the last component of **argv[0]** set to 'rsmtp' is exactly equivalent to calling it with the option **-bS**. There are no security implications in this.

49.9 Use of %f formatting

The only use made of ‘%f’ by Exim is in formatting load average values. These are actually stored in integer variables as 1000 times the load average. Consequently, their range is limited and so therefore is the length of the converted output.

49.10 Embedded Exim path

Exim uses its own path name, which is embedded in the code, only when it needs to re-exec in order to regain root privilege. Therefore it is not root when it does so. If some bug allowed the path to get overwritten, it would lead to an arbitrary program’s being run as exim, not as root. If there’s still paranoia about this, two separate copies of the name could be kept, or a checksum could be applied to the global data.

49.11 Use of sprintf()

A large number of occurrences of ‘sprintf’ in the code are actually calls to **string_sprintf()**, a function which returns the result in malloc’d store. The intermediate formatting is done into a large fixed buffer by a function that runs through the format string itself, and checks the length of each conversion before performing it, thus preventing buffer overruns. [This was not true before Exim version 1.70.]

The remaining uses of **sprintf()** happen in controlled circumstances where the output buffer is known to be sufficiently long to contain the converted string.

49.12 Use of debug_printf() and log_write()

Arbitrary strings are passed to both these functions, but they do their formatting by calling the function **string_vformat()**, which runs through the format string itself, and checks the length of each conversion. [This was not true before Exim version 1.70.]

49.13 Use of strcat() and strcpy()

These are used only in cases where the output buffer is known to be large enough to hold the result.

50. Format of spool files

A message on Exim's spool consists of two files, whose names are the message id followed by -D and -H, respectively. The data portion of the message is kept in the -D file on its own. The message's 'envelope', status, and headers are all kept in the -H file, whose format is described in this chapter. Each of these two files contains the final component of its own name as its first line. This is insurance against disc crashes where the directory is lost but the files themselves are recoverable.

Files whose names end with -J may also be seen in the spool directory. These are journal files, used to record addresses to which the message has been delivered during the course of a delivery run. At the end of the run, the -H file is updated, and the -J file is deleted.

The second line of the header file contains the login id of the process that called Exim to create the file, followed by the numerical uid and gid. For a locally generated message, this is normally the user who sent the message. For an external message, the user is either root or exim.

The third line of the file contains the address of the message's sender as transmitted in the 'envelope', contained in angle brackets. In the case of incoming SMTP mail, this is the address given in the MAIL command. For locally generated mail, the sender address is created by Exim from the login of the current user and the configured **qualify_domain**, except when Exim is called by a trusted user that supplied a sender address via the **-f** option, or a leading 'From' line. The sender address is null if the message is a delivery failure report.

The fourth line contains two numbers. The first is the time that the message was received, in the form supplied by the Unix **time()** function – a number of seconds since the start of the epoch. The second number is a count of the number of messages warning of delayed delivery that have been sent to the sender.

There follow a number of optional lines, each of which starts with a hyphen when present. These can appear in any order, and are omitted when not relevant.

- **-body_linecount** <*number*>: This records the number of lines in the body of the message.
- **-deliver_firsttime**: This is written when a new message is first added to the spool. When the spool file is updated after a deferral, it is omitted.
- **-frozen** <*time*>: The message is frozen, and the freezing happened at <*time*>. No deliveries will be attempted while the message remains frozen, but the **auto_thaw** configuration option can specify a time delay after which a delivery will be attempted.
- **-helo_name** <*text*>: This records the host name as specified by a remote host in a HELO or EHLO command.
- **-host_name** <*text*>: This records the name of the remote host from which the message was received, if the host name was looked up from the IP address. It is not present if no reverse lookup was done.
- **-host_address** <*address*>: This records the IP address of the remote host from which the message was received. It is omitted for locally generated messages.
- **-ident** <*text*>: For locally submitted messages, this records the login of the originating user, unless it was a trusted user and the **-oMt** option was used to specify an ident value. For messages received over TCP/IP, this records the ident string supplied by the remote host.
- **-interface_address** <*address*>: This records the IP address of the local interface through which a message was received from a remote host. It is omitted for locally generate d messages.
- **-local**: The message is from a local sender.
- **-localerror**: The message is a locally-generated delivery error report.

- **-manual_thaw**: The message was frozen but has been thawed manually, that is, by an explicit Exim command rather than via the auto-thaw process.
- **-received_protocol**: This records the value of the **$received_protocol** variable, which contains the name of the protocol by which the message was received.
- **-resent**: The message contains **Resent-** headers, so the alternative set of header names is to be used (see RFC 822).
- **-user_null_sender**: The message was received from an unprivileged user with the **-f** option specifying '<>' as the sender.

Following the options are those addresses to which the message is not to be delivered. This set of addresses is initialized from the command line when the **-t** option is used and **extract_addresses_remove_arguments** is set; otherwise it starts out empty. Whenever a successful delivery is made, the address is added to this set. The addresses are kept internally as a balanced binary tree, and it is a representation of that tree which is written to the spool file. If an address is expanded via an alias or forward file, the original address is added to the tree when deliveries to all its child addresses are completed.

If the tree is empty, there is a single line in the spool file containing just the text 'XX'. Otherwise, each line consists of two letters, which are either Y or N, followed by an address. The address is the value for the node of the tree, and the letters indicate whether the node has a left branch and/or a right branch attached to it, respectively. If branches exist, they immediately follow. Here is an example of a three-node tree:

```
YY darcy@austen.fict.book
NN alice@wonderland.fict.book
NN editor@thesaurus.ref.book
```

After the non-recipients tree, there is a list of the message's recipients. This is a simple list, preceded by a count. It includes all the original recipients of the message, including those to whom the message has already been delivered. In the simplest case, the list contains one address per line. For example:

```
4
editor@thesaurus.ref.book
darcy@austen.fict.book
rdo@foundation
alice@wonderland.fict.book
```

However, when a child address has been added to the top-level addresses as a result of the use of the **one_time** option on an **aliasfile** or **forwardfile** director, each line is of the following form:

<*top-level address*> <*flags number*>`,`<*parent number*>`,0`

The flags at present contain only one bit, which is set for **one_time** addresses. It indicates that <*parent number*> is the offset in the recipients list of the original parent of the address. The third number of the trio is for future expansion and is currently always zero. A blank line separates the envelope and status information from the headers which follow. A header may occupy several lines of the file, and to save effort when reading it in, each header is preceded by a number and an identifying character. The number is the number of characters in the header, including any embedded newlines and the terminating newline. The character is one of the following:

<*blank*>	header in which Exim has no special interest
`B`	**Bcc:** header
`C`	**Cc:** header
`F`	**From:** header
`I`	**Message-id:** header
`P`	**Received:** header – P for ‘postmark’
`R`	**Reply-to:** header
`S`	**Sender:** header
`T`	**To:** header
`*`	replaced or deleted header

Deleted or replaced (rewritten) headers remain in the spool file for debugging purposes. They are not transmitted when the message is delivered. When **Resent-** headers are present, it is those headers that have the appropriate flags. Here is a typical set of headers:

```
111P Received: by hobbit.fict.book with local (Exim 0.17 #8)
        id E0tHplY-0000mG-00; Tue, 21 Nov 1995 10:17:32 +0000
049  Message-Id: <E0tHplY-0000mG-00@hobbit.fict.book>
038* X-rewrote-sender: bb@hobbit.fict.book
042* From: Bilbo Baggins <bb@hobbit.fict.book>
049F From: Bilbo Baggins <B.Baggins@hobbit.fict.book>
099* To: alice@wonderland.fict.book, rdo@foundation,
 darcy@austen.fict.book, editor@thesaurus.ref.book
109T To: alice@wonderland.fict.book, rdo@foundation.fict.book,
 darcy@austen.fict.book, editor@thesaurus.ref.book
038  Date: Tue, 21 Nov 1995 10:17:32 +0000
```

The asterisked headers indicate that the envelope sender, **From:** header, and **To:** header have been rewritten, the last one because routing expanded the unqualified domain **foundation**.

51. Adding new drivers or lookup types

The following actions have to be taken in order to add a new director, router, transport or lookup type to Exim:

(1) Choose a name for the driver or lookup type that does not conflict with any existing name; I will use ‘newdriver’ in what follows.

(2) Add to **src/EDITME** the line

```
<type>_NEWDRIVER=yes
```

where <*type*> is DIRECTOR, ROUTER, TRANSPORT or LOOKUP. If the code is not to be included in the binary by default, comment this line out. You should also add any relevant comments about the driver or lookup type.

(3) Add to **src/config.h.defaults** the line

```
#define <type>_NEWDRIVER
```

(4) Edit **src/drtables.c**, adding conditional code to pull in the private header and create a table entry as is done for all the other drivers and lookup types.

(5) Edit **Makefile** in the appropriate sub-directory (**src/directors**, **src/routers**, **src/transports**, or **src/lookups**); add a line for the new driver or lookup type and add it to the definition of OBJ.

(6) Create **newdriver.h** and **newdriver.c** in the appropriate sub-directory of **src**.

(7) Edit **scripts/MakeLinks** and add commands to link the **.h** and **.c** files as for other drivers and lookups.

Then all you need to do is write the code! A good way to start is to make a proforma by copying an existing module of the same type, globally changing all occurrences of the name, and cutting out most of the code. Note that any options you create must be listed in alphabetical order, because the tables are searched using a binary chop procedure.

There is a **README** file in each of the sub-directories of **src** describing the interface that is expected.

Index