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 3QH
United Kingdom

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

Edition for Exim 4.20, May 2003

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 hosts that are running Unix or Unix-like operating systems. It was designed on the assumption that it would be run on hosts that are permanently connected to the Internet. However, it can be used on intermittently connected hosts with suitable configuration adjustments.

Configuration files currently exist for the following operating systems: AIX, BSD/OS (aka BSDI), Darwin (Mac OS X), DGUX, FreeBSD, GNU/Hurd, GNU/Linux, HI-OSF (Hitachi), HP-UX, IRIX, MIPS RISCOS, NetBSD, OpenBSD, QNX, SCO, SCO SVR4.2 (aka UNIX-SV), Solaris (aka SunOS5), SunOS4, Tru64-Unix (formerly Digital UNIX, formerly DEC-OSF1), Ultrix, and Unixware. However, code is not available for determining system load averages under Ultrix.

There are also configuration files for compiling Exim in the Cygwin environment that can be installed on systems running Windows. However, this document does not contain any information about running Exim in the Cygwin environment.

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**.

The use, supply or promotion of Exim for the purpose of sending bulk, unsolicited electronic mail is incompatible with the basic aims of the program, which revolve around the free provision of a service that enhances the quality of personal communications. The author of Exim regards indiscriminate mass-mailing as an antisocial, irresponsible abuse of the Internet.

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 MTA. Many of the ideas and user interfaces were originally taken from Smail 3, though the actual code of Exim is entirely new, and has developed far beyond the initial concept.

Many 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. The distribution now contains a file called **ACKNOWLEDGMENTS**, in which I have started recording the names of contributors.

1.1 Exim documentation

This edition of the Exim specification applies to version 4.20 of Exim. Substantive changes from the 4.10 edition are marked by bars in the right-hand margin in the PostScript, PDF, and plain text versions of the document, and by green text in the HTML version, as shown by this paragraph. Changes are not marked in the Texinfo version, because Texinfo doesn't support change bars. Minor corrections and rewordings are not marked.

This document is very much a reference manual; it is not a tutorial. The reader is expected to have some familiarity with the SMTP mail transfer protocol and with general Unix system administration. Although there are some discussions and examples in places, the information is mostly organized in a way that makes it easy to look up, rather than in a natural order for sequential reading. Furthermore, the manual aims to cover every aspect of Exim in detail, including a number of rarely-used, special-purpose features that are unlikely to be of very wide interest.

An 'easier' discussion of Exim which provides more in-depth explanatory, introductory, and tutorial material can be found in *The Exim SMTP Mail Server*, published by UIT Cambridge. This book also contains a chapter that gives a general introduction to SMTP and Internet mail. Inevitably, however, the book is unlikely to be fully up-to-date with the latest release of Exim. (Note that the earlier book about Exim, published by O'Reilly, covers Exim 3, and many things have changed in Exim 4.)

As the program develops, there may be features in newer versions 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, specifications of new features that are not yet in this manual are placed in the file **doc/NewStuff** in the Exim distribution. All changes to the program (whether new features, bug fixes, or other kinds of change) are noted briefly in the file called **doc/ChangeLog**.

This specification itself is available as an ASCII file in **doc/spec.txt** so that it can easily be searched with a text editor. Other files in the **doc** directory are:

OptionLists.txt	list of all options in alphabetical order
dbm.discuss.txt	discussion about DBM libraries
exim.8	a man page of Exim's command line options
filter.txt	specification of the filter language
pcrepattern.txt	specification of PCRE regular expressions
pcretest.txt	specification of the PCRE testing program
Exim3.upgrade	upgrade notes from release 2 to release 3
Exim4.upgrade	upgrade notes from release 3 to release 4

The main specification and the specification of the filtering language are also available in other formats (HTML, PostScript, PDF, and Texinfo). Section 1.5 below tells you how to get hold of these.

1.2 FTP and web sites, and mailing list

The primary distribution site for Exim is an FTP site, whose contents are described in *Where to find the Exim distribution* below. In addition, there is a web site at **http://www.exim.org** by courtesy of Energis Squared, formerly Planet Online Ltd, who are situated in the UK. The site is mirrored in a number of other countries; links to the mirrors are listed on the home page. The web site contains the Exim distribution, and you can also find the documentation and the FAQ online there, as well as other relevant material.

Energis Squared also provide resources for the following mailing lists:

exim-users@exim.org	general discussion list
exim-announce@exim.org	moderated, low volume announcements list
pop-imap@exim.org	discussion of POP/IMAP issues

You can subscribe to these lists, change your existing subscriptions, and view or search the archives via the mailing lists link on the Exim home page. The *exim-users* mailing list is also forwarded to **http://www.egroups.com/list/exim-users**, which is another archiving system with searching capabilities.

1.3 Exim training

From time to time (approximately annually at the time of writing), lecture-based training courses are run by the author of Exim in Cambridge, UK. The web site **http://www-tus.csx.cam.ac.uk/courses/exim/** has details of any such courses that are planned.

1.4 Bug reports

Reports of obvious bugs should be emailed to *bugs@exim.org*. However, if you are unsure whether some behaviour is a bug or not, the best thing to do is to post a message to the *exim-users* mailing list and have it discussed.

1.5 Where to find the Exim distribution

The master ftp site for the Exim distribution is

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim

Within that directory there are subdirectories called **exim3** (for previous Exim 3 distributions), **exim4** (for the latest Exim 4 distributions), and **Testing** for occasional testing versions. Those mirror sites that I know about are listed in the file

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/Mirrors

In the **exim4** subdirectory, the current release can always be found in files called

exim-*n.nn***.tar.gz**
exim-*n.nn***.tar.bz2**

where *n.nn* is the highest such version number in the directory. The two files contain identical data; the only difference is the type of compression. The **.bz2** file is usually a lot smaller than the **.gz** file. The distributions are signed with Philip Hazel's GPG key, and the signatures are in:

exim-*n.nn***.tar.gz.sig**
exim-*n.nn***.tar.bz2.sig**

When there is only a small amount of change from one release 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.csx.cam.ac.uk/pub/software/email/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 inside the **exim4** directory of the FTP site:

exim-html-*n.nn***.tar.gz**
exim-pdf-*n.nn***.tar.gz**
exim-postscript-*n.nn***.tar.gz**
exim-texinfo-*n.nn***.tar.gz**

These tar files contain only the **doc** directory, not the complete distribution, and are also available in **.bz2** as well as **.gz** forms.

The FAQ is available for downloading in two different formats from

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/exim4/FAQ.txt.gz
ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/exim4/FAQ.html.tar.gz

The first of these is a single ASCII file that can be searched with a text editor. The second is a directory of HTML files, normally accessed by starting at **index.html**. The HTML version of the FAQ (which is also included in the HTML documentation tarbundle) includes a keyword-in-context index, which is often the most convenient way of finding your way around.

1.6 Wish list

A wish list is maintained, containing ideas for new features that have been submitted. From time to time the file is exported to the ftp site:

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/exim4/WishList

Items are removed from the list if they get implemented.

1.7 Contributed material

At the ftp site, there is a directory called

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/exim4/Contrib/

which contains miscellaneous files contributed to the Exim community by Exim users. There is also a collection of contributed configuration examples in

ftp://ftp.csx.cam.ac.uk/pub/software/email/exim/exim4/config.samples.tar.gz

These samples are referenced from the FAQ.

1.8 Limitations

- Exim is designed for use as an Internet MTA, and therefore handles addresses in RFC 2822 domain format only. It cannot handle UUCP 'bang paths', though simple two-component bang paths can be converted by a straightforward rewriting configuration. This restriction does not prevent Exim from being interfaced to UUCP as a transport mechanism, provided that domain addresses are used.
- 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 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.
- Exim is not designed for storing mail for dial-in hosts. When the volumes of such mail are large, it is better to get the messages 'delivered' into files (that is, off Exim's queue) and subsequently passed on to the dial-in hosts by other means.

1.9 Run time configuration

Exim's run time 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 online installations is provided in the distribution, and is described in chapter 7 below.

1.10 Calling interface

Like many MTAs, Exim has adopted the Sendmail command line interface so that it can be a straight replacement for **/usr/lib/sendmail** or **/usr/sbin/sendmail** when sending mail, but you do not need to know anything about Sendmail in order to run Exim. For actions other than sending messages, Sendmail-compatible options also exist, but those that produce output (for example, **-bp**, which lists the messages on the queue) do so in Exim's own format. There are also some additional options that are compatible with Smail 3, and some further options that are new to Exim. Chapter 5 documents all Exim's command line options. This information is automatically made into the man page which forms part of the Exim 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 which contains a menu interface to Exim's command line administration options.

1.11 Terminology

The *body* of a message is the actual data that the sender wants to transmit. It is the last part of a message, and is separated from the *header* (see below) by a blank line.

When a message cannot be delivered, it is normally returned to the sender in a delivery failure message. The term *bounce* is commonly used for this action, and the error reports are often called *bounce messages*. This is a convenient shorthand for 'delivery failure error report'. Such messages have an empty sender address in the message's *envelope* (see below) to ensure that they cannot themselves give rise to further bounce messages.

The term *default* appears frequently in this manual. It is used to qualify a value which is used in the absence of any setting in the configuration. It may also qualify an action which is taken unless a configuration setting specifies otherwise.

The term *defer* is used when the delivery of a message to a specific destination cannot immediately take place for some reason (a remote host may be down, or a user's local mailbox may be full). Such deliveries are *deferred* until a later time.

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

A message in transit has an associated *envelope*, as well as a header and a body. The envelope contains a sender address (to which bounce messages should be delivered), and any number of recipient addresses. References to the sender or the recipients of a message usually mean the addresses in the envelope. An MTA uses these addresses for delivery, and for returning bounce messages, not the addresses that appear in the header lines.

The *header* of a message is the first part of a message's text, consisting of a number of lines, each of which has a name such as *From:*, *To:*, *Subject:*, etc. Long header lines can be split over several text lines by indenting the continuations. The header is separated from the body by a blank line.

The term *local part*, which is taken from RFC 2822, 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 terms *local delivery* and *remote delivery* are used to distinguish delivery to a file or a pipe on the local host from delivery by SMTP over TCP/IP to a remote host.

Return path is another name that is used for the sender address in a message's envelope.

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.

The term *spool directory* is used for a directory in which Exim keeps the messages on its queue – that is, those that it is in the process of delivering. This should not be confused with the directory in which local mailboxes are stored, which is called a 'spool directory' by some people. In the Exim documentation, 'spool' is always used in the first sense.

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 © 2003 University of Cambridge. The source is distributed in the directory **src/pcre**. However, this is a cut-down version of PCRE. If you want to use the PCRE library in other programs, you should obtain and install the full version from **ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre**.

- 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:

Copyright © 1998 Nigel Metheringham, Planet Online Ltd

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This code implements Dan Bernstein's Constant DataBase (cdb) spec. Information, the spec and sample code for cdb can be obtained from **http://www.pobox.com/~djb/cdb.html**. This implementation borrows some code from Dan Bernstein's implementation (which has no license restrictions applied to it).

The implementation is completely contained within the code of Exim. It does not link against an external cdb library.

- Client support for Microsoft's *Secure Password Authentication* is provided by code contributed by Marc Prud'hommeaux. This includes code taken from the Samba project, which is released under the Gnu GPL.

- Support for calling the Cyrus *pwcheck* daemon is provided by code taken from the Cyrus-SASL library and adapted by Alexander S. Sabourenkov. The permission notice appears below, in accordance with the conditions expressed therein.

Copyright © 2001 Carnegie Mellon University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name 'Carnegie Mellon University' must not be used to endorse or promote products derived from this software without prior written permission. For permission or any other legal details, please contact

   Office of Technology Transfer
   Carnegie Mellon University
   5000 Forbes Avenue
   Pittsburgh, PA 15213-3890
   (412) 268-4387, fax: (412) 268-7395
   tech-transfer@andrew.cmu.edu

4. Redistributions of any form whatsoever must retain the following acknowledgment:
   *This product includes software developed by Computing Services at Carnegie Mellon University (***http://www.cmu.edu/computing/***).*

CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- 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.

Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

3. How Exim receives and delivers mail

3.1 Overall 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 Policy control

Policy controls are now an important feature of MTAs that are connected to the Internet. Perhaps their most important job is to stop MTAs being abused as 'open relays' by misguided individuals who send out vast amounts of unsolicited junk, and want to disguise its source. Exim provides flexible facilities for specifying policy controls on incoming mail:

- Exim 4 (unlike previous versions of Exim) implements policy controls on incoming SMTP mail by means of *Access Control Lists* (ACLs). Each list is a series of statements that may either grant or deny access. ACLs can be used at several places in the SMTP dialogue while receiving a message. However, the most common places are after each RCPT command, and at the very end of the message. The sysadmin can specify conditions for accepting or rejecting individual recipients or the entire message (see chapter 37), respectively, at these two points. Denial of access results in an SMTP error code.
- An ACL is also available for locally generated, non-SMTP messages. In this case, the only available actions are to accept or deny the entire message.
- When a message has been received, either from a remote host or from the local host, but before the final acknowledgement has been sent, a locally supplied C function called *local_scan()* can be run to inspect the message and decide whether to accept it or not (see chapter 38). If the message is accepted, the list of recipients can be modified by the function.
- After a message has been accepted, a further checking mechanism is available in the form of the *system filter* (see chapter 39). This runs at the start of every delivery process.

3.3 Message identification

Every message handled by Exim is given a *message id* which is sixteen characters long. It is divided into three parts, separated by hyphens, for example `16VDhn-0001bo-D3`. Each part is a sequence of letters and digits, normally encoding numbers in base 62. However, in the Darwin operating system (Mac OS X) and when Exim is compiled to run under Cygwin, base 36 (avoiding the use of lower case letters) is used instead, because the message id is used to construct file names, and the names of files in those systems are not case-sensitive.

The detail of the contents of the message id have changed as Exim has evolved. Earlier versions relied on the operating system not re-using a process id (pid) within one second. On modern operating systems, this assumption can no longer be made, so the algorithm had to be changed. To retain backward compatibility, the format of the message id was retained, which is why the following rules are somewhat eccentric:

- The first six characters of the message id are the time at which the message started to be received, to a granularity of one second. That is, this field contains the number of seconds since the start of the epoch (the normal Unix way of representing the date and time of day).
- After the first hyphen, the next six characters are the id of the process that received the message.
- There are two different possibilities for the final two characters:

(a) If **localhost_number** is not set, this value is the fractional part of the time of reception, normally in units of 1/2000 of a second, but for systems that must use base 36 instead of base 62 (because of case-insensitive file systems), the units are 1/1000 of a second.

(b) If **localhost_number** is set, it is multiplied by 200 (100) and added to the fractional part of the time, which in this case is in units of 1/200 (1/100) of a second.

After a message has been received, Exim waits for the clock to tick at the appropriate resolution before proceeding, so that if another message is received by the same process, or by another process with the same (re-used) pid, it is guaranteed that the time will be different. In most cases, the clock will already have ticked while the message was being received.

3.4 Receiving mail

The only way Exim can receive mail from a remote host is using SMTP over TCP/IP, in which case the sender and recipient addresses are tranferred using SMTP commands. However, from a locally running process (such as a user's MUA), there are several possibilities:

- If the process runs Exim with the **-bm** option, the message is read non-interactively (usually via a pipe), with the recipients taken from the command line, or from the body of the message if **-t** is also used.
- If the process runs Exim with the **-bS** option, the message is also read non-interactively, but in this case the recipients are listed at the start of the message in a series of SMTP RCPT commands, terminated by a DATA command. This is so-called 'batch SMTP' format, but it isn't really SMTP. The SMTP commands are just another way of passing envelope addresses in a non-interactive submission.
- If the process runs Exim with the **-bs** option, the message is read interactively, using the SMTP protocol. A two-way pipe is normally used for passing data between the local process and the Exim process. This is 'real' SMTP and is handled in the same way as SMTP over TCP/IP. For example, the ACLs for SMTP commands are used for this form of submission.
- A local process may also make a TCP/IP call to the host's loopback address (127.0.0.1) or any other of its IP addresses. When receiving messages, Exim does not treat the loopback address specially. It treats all such connections in the same way as connections from other hosts.

In the three cases that do not involve TCP/IP, the sender address is constructed from the login name of the user that called Exim and a default qualification domain (which can be set by the **qualify_domain** configuration option). For local or batch SMTP, a sender address that is passed using the SMTP MAIL command is ignored. However, the system administrator may allow certain users ('trusted users') to specify a different sender address unconditionally, or all users to specify certain forms of different sender address. The **-f** option or the SMTP MAIL command is used to specify these different addresses. See section 5.2 for details of trusted users, and the **untrusted_set_sender** option for a way of allowing untrusted users to change sender addresses.

Messages received by either of the non-interactive mechanisms are subject to checking by the non-SMTP ACL, if one is defined. Messages received using SMTP (either over TCP/IP, or interacting with a local process) can be checked by a number of ACLs that operate at different times during the SMTP session. Either individual recipients, or the entire message, can be rejected if local policy requirements are not met. The *local_scan()* function (see chapter 38) is run for all incoming messages.

Exim can be configured not to start a delivery process when a message is received; this can be unconditional, or depend on the number of incoming SMTP connections or the system load. In these situations, new messages wait on the queue until a queue runner process picks them up. However, in standard configurations under normal conditions, delivery is started as soon as a message is received.

3.5 Handling an incoming message

When Exim accepts a message, it writes two files in its spool directory. The first contains the envelope information, the current status of the message, and the header lines, and the second contains the body of the message. The names of the two spool files consist of the message id, followed by `-H` for the file containing the envelope and header, and `-D` for the data file.

By default all these message 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, the **split_spool_directory** option can be used. This causes Exim to split up the input files into 62 sub-directories whose names are single letters or digits.

The envelope information consists of the address of the message's sender and the addresses of the recipients. This information is entirely separate from any addresses contained in the header lines. 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 48.

Address rewriting that is specified in the rewrite section of the configuration (see chapter 30) is done once and for all on incoming addresses, both in the header lines and the envelope, at the time the message is accepted. If during the course of delivery additional addresses are generated (for example, via aliasing), these new addresses are rewritten as soon as they are generated. At the time a message is actually delivered (transported) further rewriting can take place; because this is a transport option, it can be different for different forms of delivery. It is also possible to specify the addition or removal of certain header lines at the time the message is delivered (see chapters 14 and 23).

3.6 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 a bounce message to be sent.

There is an option called **auto_thaw**, 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 (bounce messages) that reach this timeout are discarded. There is also an option called **timeout_frozen_after**, which discards frozen messages after a certain time.

While Exim is working on a message, it writes information about each delivery attempt to the main log file. This includes successful, unsuccessful, and delayed deliveries for each recipient (see chapter 44). The log lines are also written to a separate *message log* file for each message. These logs are solely for the benefit of the administrator, and are normally deleted along with the spool files when processing of a message is complete. The use of individual message logs can be disabled by setting **no_message_logs**; this might give an improvement in performance on very busy systems.

All the information Exim itself needs to set up a delivery is kept in the first spool file, along with the header lines. 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 (the `-H` 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.

3.7 Processing an address for delivery

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

Each driver that is specified in the run time configuration is an *instance* of that particular driver type. Multiple instances are allowed; for example, you can set up several different **smtp** transports, each with different option values that might specify different ports or different timeouts. Each instance has its own identifying name. In what follows we will normally use the instance name when discussing one particular instance (that is, one specific configuration of the driver), and the generic driver name when discussing the driver's features in general.

A *router* is a driver that operates on an address, either determining how its delivery should happen, by routing it to a specific transport, or converting the address into one or more new addresses (for example, via an alias file). A router may also explicitly choose to fail an address, causing it to be bounced.

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, whereas for a *remote* transport the destination is some other host. A message is passed to a specific transport as a result of successful routing. If a message has several recipients, it may be passed to a number of different transports.

An address is processed by passing it to each configured router instance in turn, subject to certain preconditions, until a router accepts the address or specifies that it should be bounced. We will describe this process in more detail shortly. As a simple example, the diagram below illustrates how each recipient address in a message is processed in a small configuration of three routers that are configured in various ways.

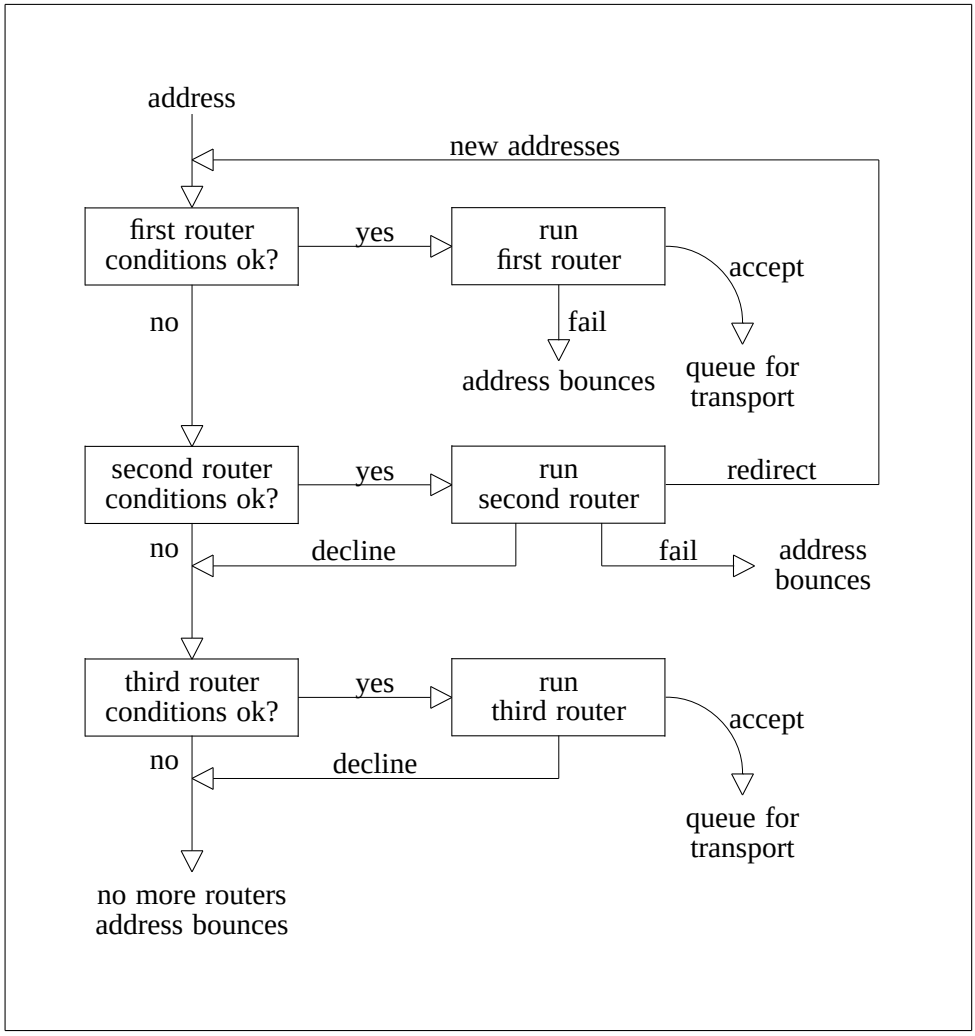
To make this a more concrete example, we'll describe it in terms of some actual routers, but remember, this is only an example. You can configure Exim's routers in many different ways, and there may be any number of routers in a configuration.

The first router that is specified in a configuration is often one that handles addresses in domains that are not recognized specially by the local host. These are typically addresses for arbitrary domains on the Internet. A precondition is set up which looks for the special domains known to the host (for example, its own domain name), and the router is run for addresses that do *not* match. Typically, this is a router that looks up domains in the DNS in order to find the hosts to which this address routes. If it succeeds, the address is queued for a suitable SMTP transport; it it does not succeed, the router is configured to fail the address.

The example pictured could be a configuration of this type. The second and third routers can only be run for addresses for which the preconditions for the first router are not met. If one of these preconditions checks the domain, the second and third routers are run only for domains that are somehow special to the local host.

The second router does redirection – also known as aliasing and forwarding. When it generates one or more new addresses from the original, each of them is routed independently from the start. Otherwise, the router may cause an address to fail, or it may simply decline to handle the address, in which case the address is passed to the next router.

The final router in many configurations is one that checks to see if the address belongs to a local mailbox. The precondition may involve a check to see if the local part is the name of a login account, or it may look up the local part in a file or a database. If its preconditions are not met, or if the router declines, we have reached the end of the routers. When this happens, the address is bounced.



Routing an address

3.8 Running an individual router

As explained in the example above, a number of preconditions are checked before running a router. If any are not met, the router is skipped, and the address is passed to the next router. When all the preconditions on a router *are* met, the router is run. What happens next depends on the outcome, which is one of the following:

- *accept*: The router accepts the address, and either queues it for a transport, or generates one or more 'child' addresses. Processing the original address ceases, unless the **unseen** option is set on the router. This option can be used to set up multiple deliveries with different routing (for example, for keeping archive copies of messages). When **unseen** is set, the address is passed to the next router. Normally, however, an *accept* return marks the end of routing.

  If child addresses are generated, Exim checks to see whether they are duplicates of any existing recipient addresses. During this check, local parts are treated as case-sensitive. Duplicate addresses are discarded. Each of the remaining child addresses is then processed independently, starting with the first router by default. It is possible to change this by setting the **redirect_router** option to specify which router to start at for child addresses. Unlike **pass_router** (see below) the router specified by **redirect_router** may be anywhere in the router configuration.

- *pass*: The router recognizes the address, but cannot handle it itself. It requests that the address be passed to another router. By default the address is passed to the next router, but this can be

changed by setting the **pass_router** option. However, (unlike **redirect_router**) the named router must be below the current router (to avoid loops).

- *decline*: The router declines to accept the address because it does not recognize it at all. By default, the address is passed to the next router, but this can be prevented by setting the **no_more** option. When **no_more** is set, all the remaining routers are skipped.
- *fail*: The router determines that the address should fail, and queues it for the generation of a bounce message. There is no further processing of the original address unless **unseen** is set on the router.
- *defer*: The router cannot handle the address at the present time. (A database may be offline, or a DNS lookup may have timed out.) No further processing of the address happens in this delivery attempt. It is tried again next time the message is considered for delivery.
- *error*: There is some error in the router (for example, a syntax error in its configuration). The action is as for defer.

If an address reaches the end of the routers without having been accepted by any of them, it is bounced as unrouteable. The default error message in this situation is 'unrouteable address', but you can set your own message by making use of the **cannot_route_message** option. This can be set for any router; the value from the last router that 'saw' the address is used.

Sometimes while routing you want to fail a delivery when some conditions are met but others are not, instead of passing the address on for further routing. You can do this by having a second router that explicitly fails the delivery when the relevant conditions are met. The **redirect** router has a 'fail' facility for this purpose.

3.9 Router preconditions

The preconditions that are tested for each router are listed below, in the order in which they are tested. The individual configuration options are described in more detail in chapter 14.

- The **local_part_prefix** and **local_part_suffix** options can specify that the local parts handled by the router may or must have certain prefixes and/or suffixes. If a mandatory affix (prefix or suffix) is not present, the router is skipped. These conditions are tested first. When an affix is present, it is removed from the local part before further processing, including the evaluation of any other conditions.
- Routers can be designated for use only when not verifying an address, that is, only when routing it for delivery (or testing its delivery routing). If the **verify** option is set false, the router is skipped when Exim is verifying an address.
- If the **address_test** option is set false, the router is skipped when Exim is run with the **-bt** option to test an address routing. This can be helpful when the first router sends all new messages to a scanner of some sort; it makes it possible to use **-bt** to test subsequent delivery routing without having to simulate the effect of the scanner.
- Routers can be designated for use only when verifying an address, as opposed to routing it for delivery. The **verify_only** option controls this.
- Certain routers can be explicitly skipped when running the routers to check an address given in the SMTP EXPN command (see the **expn** option).
- If the **domains** option is set, the domain of the address must be in the set of domains that it defines.
- If the **local_parts** option is set, the local part of the address must be in the set of local parts that it defines. If **local_part_prefix** or **local_part_suffix** is in use, the prefix or suffix is removed from the local part before this check. If you want to do precondition tests on local parts that include affixes, you can do so by using a **condition** option (see below) that uses the variables **$local_part**, **$local_part_prefix**, and **$local_part_suffix** as necessary.

- If the **check_local_user** option is set, the local part must be the name of an account on the local host.
- If the **router_home_directory** option is set, it is expanded at this point, because it overrides the value of **$home**. If this expansion were left till later, the value of **$home** as set by **check_local_user** would be used in subsequent tests. Having two different values of **$home** in the same router could lead to confusion.
- If the **senders** option is set, the envelope sender address must be in the set of addresses that it defines.
- If the **require_files** option is set, the existence or non-existence of specified files is tested.
- If the **condition** option is set, it is evaluated and tested. This option uses an expanded string to allow you to set up your own custom preconditions. Expanded strings are described in chapter 11.

Note that **require_files** comes near the end of the list, so you cannot use it to check for the existence of a file in which to lookup up a domain, local part, or sender. However, as these options are all expanded, you can use the **exists** expansion condition to make such tests within each condition. The **require_files** option is intended for checking files that the router may be going to use internally, or which are needed by a specific transport (for example, **.procmailrc**).

3.10 Delivery in detail

When a message is to be delivered, the sequence of events is 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. The format of the filter file is the same as for user filter files, described in the separate document entitled *Exim's interface to mail filtering*. Some additional features are available in system filters – see chapter 39 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 the first run of the system filter.
- Each recipient address is offered to each configured router in turn, subject to its preconditions, until one is able to handle it. If no router can handle the address, that is, if they all decline, the address is failed. Because routers can be targeted at particular domains, several locally handled domains can be processed entirely independently of each other.
- A router that accepts an address may set up a local or a remote transport for it. However, the transport is not run at this time. Instead, the address is placed on a queue for the particular transport, to be run later. Alternatively, the router 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 router ignores any address which has an identically-named ancestor that was processed by itself.
- When all the routing has been 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 a time, 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. Remote deliveries also run in separate processes, normally under a uid that is private to Exim ('the Exim user'), but in this case, several remote deliveries can be run in parallel. The maximum number of simultaneous remote deliveries for any one message is set by the **remote_max_parallel** option.

- When it encounters a local delivery during a queue run, Exim checks its retry database to see if there has been a previous temporary delivery failure for the address before running the local transport. If there was a previous failure, Exim 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. Local deliveries are always attempted when delivery immediately follows message reception, 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).
- 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 temporary 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 permanent errors, a bounce 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 bounce 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. Delivery of these addresses is said to be *deferred*.
- When all the recipient addresses have either been delivered or bounced, handling of the message is complete. The spool files and message log are deleted, though the message log can optionally be preserved if required.

3.11 Retry mechanism

Exim's mechanism for retrying messages that fail to get delivered at the first attempt is the queue runner process. You must either run an Exim daemon that uses the **-q** option with a time interval to start queue runners at regular intervals, or use some other means (such as *cron*) to start them. If you do not arrange for queue runners to be run, messages that fail temporarily at the first attempt will remain on your queue for ever. A queue runner process works it way through the queue, one message at a time, trying each delivery that has passed its retry time. You can run several queue runners at once.

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 bounce message. If no retry rules are set for a particular host, address, and error combination, no retries are attempted, and temporary errors are treated as permanent.

3.12 Temporary delivery failure

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. Temporary failures may be detected during routing as well as during the transport stage of delivery. Local deliveries may 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.

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.

3.13 Permanent delivery failure

When a message cannot be delivered to some or all of its intended recipients, a bounce message is generated. Temporary delivery failures turn into permanent errors when their timeout expires. All the addresses that fail in a given delivery attempt are listed in a single message. If the original 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 bounce message. The wording of bounce messages can be customized by the administrator. See chapter 40 for details.

Bounce messages contain an *X-Failed-Recipients:* header line that lists the failed addresses, for the benefit of programs that try to analyse such messages automatically.

A bounce message 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 section 41.2) it is common to direct bounce messages to the manager of the list.

3.14 Failures to deliver bounce messages

If a bounce message (either locally generated or received from a remote host) itself suffers a permanent delivery failure, the message is left on the queue, but it is frozen, awaiting the attention of an administrator. There are options which can be used to make Exim discard such failed messages, or to keep them for only a short time (see **timeout_frozen_after** and **ignore_bounce_errors_after**).

4. Building and installing Exim

4.1 Unpacking

Exim is distributed as a gzipped or bzipped tar file which, when upacked, creates a directory with the name of the current release (for example, **exim-4.20**) into which the following files are placed:

ACKNOWLEDGMENTS	contains some acknowledgments
CHANGES	contains a reference to where changes are documented
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:

Local	an empty directory for local configuration files
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

The main utility programs are contained in the **src** directory, and are built with the Exim binary. The **util** directory contains a few optional scripts that may be useful to some sites.

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

Even if you do not use any DBM files in your configuration, Exim still needs a DBM library in order to operate, because it uses indexed files for its hints databases. Unfortunately, there are a number of DBM libraries in existence, and different operating systems often have different ones installed.

If you are using Solaris, IRIX, one of the modern BSD systems, or a modern Linux distribution, the DBM configuration should happen automatically, and you may be able to ignore this section. Otherwise, you may have to learn more than you would like about DBM libraries from what follows.

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 early 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, operates on two files called **dbmfile.dir** and **dbmfile.pag**.

(2) The GNU library, *gdbm*, operates on a single file. If used via its *ndbm* compatibility interface it makes two different hard links to it with names **dbmfile.dir** and **dbmfile.pag**, but if used via its native interface, the file name is used unmodified.

(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.

(5) To complicate things further, there are several very different versions of the Berkeley DB package. Version 1.85 was stable for a very long time, releases 2.*x* and 3.*x* were current for a while, but the latest versions are now numbered 4.*x*. Maintenance of some of the earlier releases has ceased. All versions of Berkeley DB can be obtained from

    http://www.sleepycat.com/

(6) Yet another DBM library, called *tdb*, has become available from

    http://download.sourceforge.net/tdb

    It has its own interface, and also operates on a single file.

Exim and its utilities can be compiled to use any of these interfaces. In order to use any version of the Berkeley DB package in native mode, you must set USE_DB in an appropriate configuration file (typically **Local/Makefile**). For example:

```
USE_DB=yes
```

Similarly, for gdbm you set USE_GDBM, and for tdb you set USE_TDB. An error is diagnosed if you set more than one of these.

At the lowest level, the build-time configuration sets none of these options, thereby assuming an interface of type (1). However, some operating system configuration files (for example, those for the BSD operating systems and Linux) assume type (4) by setting USE_DB as their default, and the configuration files for Cygwin set USE_GDBM. Anything you set in **Local/Makefile**, however, overrides these system defaults.

As well as setting USE_DB, USE_GDBM, or USE_TDB, it may also be necessary to set DBMLIB, to cause inclusion of the appropriate library, as in one of these lines:

```
DBMLIB = -ldb
DBMLIB = -ltdb
```

Settings like that will work if the DBM library is installed in the standard place. Sometimes it is not, and the library's header file may also not be in the default path. You may need to set INCLUDE to specify where the header file is, and to specify the path to the library more fully in DBMLIB, as in this example:

```
INCLUDE=-I/usr/local/include/db-4.1
DBMLIB=/usr/local/lib/db-4.1/libdb.a
```

There is further detailed discussion about the various DBM libraries in the file **doc/dbm.discuss.txt** in the Exim distribution.

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. These descriptions are therefore not repeated here. If you are building Exim for the first time, the simplest thing to do is to copy **src/EDITME** to **Local/Makefile**, then read it and edit it appropriately.

There are three settings that you must supply, because Exim will not build without them. They are the location of the run time configuration file (CONFIGURE_FILE), the directory in which Exim binaries will be installed (BIN_DIRECTORY), and the identity of the Exim user (EXIM_USER and maybe EXIM_GROUP as well). The value of CONFIGURE_FILE can in fact be a colon-separated list of file names; Exim uses the first of them that exists.

There are a few other 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**, **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 4.9 below for details of how to do this.

4.5 Including TLS/SSL encryption support

Exim can be built to support encrypted SMTP connections, using the STARTTLS command as per RFC 2487. It can also support legacy clients that expect to start a TLS session immediately on connection to a non-standard port (see the **-tls-on-connect** command line option).

If you want to build Exim with TLS support, you must first install either the OpenSSL or GnuTLS library. There is no cryptographic code in Exim itself for implementing SSL.

If OpenSSL is installed, you should set

```
SUPPORT_TLS=yes
TLS_LIBS=-lssl -lcrypto
```

in **Local/Makefile**. You may also need to specify the locations of the OpenSSL library and include files. For example:

```
SUPPORT_TLS=yes
TLS_LIBS=-L/usr/local/openssl/lib -lssl -lcrypto
TLS_INCLUDE=-I/usr/local/openssl/include/
```

If GnuTLS is installed, you should set

```
SUPPORT_TLS=yes
USE_GNUTLS=yes
TLS_LIBS=-lgnutls -ltasn1 -lgcrypt
```

in **Local/Makefile**, and again you may need to specify the locations of the library and include files. For example:

```
SUPPORT_TLS=yes
USE_GNUTLS=yes
TLS_LIBS=-L/usr/gnu/lib -lgnutls -ltasn1 -lgcrypt
TLS_INCLUDE=-I/usr/gnu/include
```

You do not need to set TLS_INCLUDE if the relevant directory is already specified in INCLUDE. Details of how to configure Exim to make use of TLS are given in chapter 36.

4.6 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_EXIM. For example, if *tcpwrappers* is installed in **/usr/local**, you might have

```
USE_TCP_WRAPPERS=yes
CFLAGS=-O -I/usr/local/include
EXTRALIBS_EXIM=-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  192.168.1.  .friendly.domain.example
```

in your **/etc/hosts.allow** file allows connections from the local host, from the subnet 192.168.1.0/24, and from all hosts in *friendly.domain.example*. All other connections are denied. Consult the *tcpwrappers* documentation for further details.

4.7 Including support for IPv6

Exim contains code for use on systems that have IPv6 support. 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 on systems where the IPv6 support is not fully integrated into the normal include and library files.

IPv6 is still changing rapidly. Two different types of DNS record for handling IPv6 addresses have been defined. AAAA records are already in use, and are currently seen as the 'mainstream', but another record type called A6 is being argued about. Its status is currently 'experimental'. Exim has support for A6 records, but this is included only if you set SUPPORT_A6=YES in **Local/Makefile**.

4.8 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 8, the directory **build-SunOS5-5.8-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 that 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*. This 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 utility programs, 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.

4.9 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 EXIM_OSTYPE or EXIM_ARCHTYPE is set, their values are used, thereby providing a means of forcing particular settings. Otherwise, the scripts try to get values from the **uname** command. If this fails, the shell variables OSTYPE and ARCHTYPE are inspected. A number of *ad hoc* transformations are then applied, to produce the standard names that Exim expects. 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, you do not need to change the original files. Instead, you should make the changes by putting the new values in an appropriate **Local** file. For example, when building Exim in many releases of the Tru64-Unix (formerly Digital UNIX, formerly DEC-OSF1) operating system, it is necessary to specify that the C compiler is called *cc* rather than *gcc*. Also, the compiler must be called with the option **-std1**, to make it recognize some of the features of Standard C that Exim uses. (Most other compilers recognize Standard C by default.) To to this, you should create a file called **Local/Makefile-OSF1** containing the lines

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

If you are compiling for just one operating system, it may be easier to put these lines directly into **Local/Makefile**.

Keeping all your local configuration settings separate from the distributed files makes it easy to transfer them to new versions of Exim simply by copying the contents of the **Local** directory.

Exim contains support for doing LDAP, NIS, NIS+, and other kinds of file lookup, 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
```

and similar settings apply to the other lookup types. They are all listed in **src/EDITME**. In most cases the relevant include files and interface libraries need to be installed before compiling Exim. However, in the case of cdb, which is included in the binary only if

```
LOOKUP_CDB=yes
```

is set, the code is entirely contained within Exim, and no external include files or libraries are required. When a lookup type is not included in the binary, attempts to configure Exim to use it cause run time configuration errors.

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 12.

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. Exim itself makes no use of X11, but if you are compiling the Exim monitor, the X11 libraries must be available. The following three variables are set in **OS/Makefile-Default**:

```
X11=/usr/X11R6
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. In contrast, EXTRALIBS_EXIM is used only on the command for linking the main Exim binary, and not for any associated utilities. 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.

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.

4.10 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.

4.11 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.

4.12 Installing Exim binaries and scripts

The command *make install* runs the *exim_install* script with no arguments. The script copies binaries and utility scripts into the directory whose name is specified by the BIN_DIRECTORY setting in **Local/Makefile**.

Exim's run time configuration file is named by the CONFIGURE_FILE setting in **Local/Makefile**. If this names a single file, and the file does not exist, the default configuration file **src/configure.default** is copied there by the installation script. If a run time configuration file already exists, it is left alone. If CONFIGURE_FILE is a colon-separated list, naming several alternative files, no default is installed.

One change is made to the default configuration file when it is installed: the default configuration contains a router that references a system aliases file. The path to this file is set to the value specified by SYSTEM_ALIASES_FILE in **Local/Makefile** (**/etc/aliases** by default). If the system aliases file does not exist, the installation script creates it, and outputs a comment to the user.

The created file contains no aliases, but it does contain comments about the aliases a site should normally have. Mail aliases have traditionally been kept in **/etc/aliases**. However, some operating systems are now using **/etc/mail/aliases**. You should check if yours is one of these, and change Exim's configuration if necessary.

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. System aliases and **.forward** files in users' home directories are supported, but no NIS or NIS+ support is configured. Domains other than the name of the local host are routed using the DNS, with delivery over SMTP.

The install script copies files only if they are newer than the files they are going to replace. The Exim binary is required to be owned by root and have the *setuid* bit set, for normal configurations. Therefore, you must run *make install* as root so that it can set up the Exim binary in this way. However, in some special situations (for example, if a host is doing no local deliveries) it may be possible to run Exim without making the binary setuid root (see chapter 47 for details).

It is possible to install Exim for special purposes (such as building a binary distribution) in a private part of the file system. You can do this by a command such as

```
make DESTDIR=/some/directory/ install
```

This has the effect of pre-pending the specified directory to all the file paths, except the name of the system aliases file that appears in the default configuration. (If a default alias file is created, its name *is* modified.) For backwards compatibility, ROOT is used if DESTDIR is not set, but this usage is deprecated.

Running *make install* does not copy the Exim 4 conversion script *convert4r4*, or the *pcretest* test program. You will probably run the first of these only once (if you are upgrading from Exim 3), and the second isn't really part of Exim. None of the documentation files in the **doc** directory are copied, except for the info files when you have set INFO_DIRECTORY, as described in section 4.13 below.

For the utility programs, old versions are renamed by adding the suffix **.O** to their names. The Exim binary itself, however, is handled differently. It is installed under a name that includes the version number and the compile number, for example **exim-4.20-1**. The script then arranges for a symbolic link called **exim** to point to the binary. If you are updating a previous version of Exim, the script takes care to ensure that the name **exim** is never absent from the directory (as seen by other processes).

If you want to see what the *make install* will do before running it for real, you can pass the **-n** option to the installation script by this command:

```
make INSTALL_ARG=-n install
```

The contents of the variable INSTALL_ARG are passed to the installation script. You do not need to be root to run this test. Alternatively, you can run the installation script directly, but this must be from within the build directory. For example, from the top-level Exim directory you could use this command:

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

There are two other options that can be supplied to the installation script.

- **-no_chown** bypasses the call to change the owner of the installed binary to root, and the call to make it a setuid binary.
- **-no_symlink** bypasses the setting up of the symbolic link **exim** to the installed binary.

INSTALL_ARG can be used to pass these options to the script. For example:

```
make INSTALL_ARG=-no_symlink install
```

The installation script can also be given arguments specifying which files are to be copied. For example, to install just the Exim binary, and nothing else, without creating the symbolic link, you could use:

```
make INSTALL_ARG='-no_symlink exim' install
```

4.13 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 1.5).

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

4.14 Setting up the spool directory

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

4.15 Testing

Having installed Exim, you can check that the run time configuration file is syntactically valid by running the following command, which assumes that the Exim binary directory is within your PATH environment variable:

```
exim -bV
```

If there are any errors in the configuration file, Exim outputs error messages. Otherwise it outputs the version number and build date, the DBM library that is being used, and information about which drivers and other optional code modules are included in the binary. Some simple routing tests can be done by using the address testing option. For example,

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

should verify that it recognizes a local mailbox, and

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

This is a test message.
^D
```

The **-v** option causes Exim to output some verification of what it is doing. In this case you should see copies of three log lines, one for the message's arrival, one for its delivery, and one containing 'Completed'.

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. If a message is stuck on Exim's spool, you can force a delivery with debugging turned on by a command of the form

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

You must be root or an 'admin user' in order to do this. The **-d** option produces rather a lot of output, but you can cut this down to specific areas. For example, if you use **-d-all+route** only the debugging information relevant to routing is included. (See the **-d** option in chapter 5 for more details.)

One specific problem that has shown up on some sites is the inability to do local deliveries into a shared mailbox directory, because it 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 25.

One thing that cannot be tested on a system that is already running an MTA 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. The **-bh** option and the *exim_checkaccess* utility can be used to check out policy controls on incoming SMTP mail.

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 run time configuration, all other file and directory names that Exim uses can be altered, in order to keep it entirely clear of the production version.

4.16 Replacing another MTA with Exim

Building and installing Exim for the first time does not of itself put it in general use. The name by which the system's MTA is called by mail user agents is either **/usr/sbin/sendmail**, or **/usr/lib/sendmail** (depending on the operating system), and it is necessary to make this name point to the *exim* binary in order to get the user agents to pass messages to Exim. This is normally done by renaming any existing file and making **/usr/sbin/sendmail** or **/usr/lib/sendmail** a symbolic link to the *exim* binary. It is a good idea to remove any setuid privilege and executable status from the old MTA. It is then necessary to stop and restart the mailer daemon, if one is running.

Some operating systems have introduced alternative ways of switching MTAs. For example, if you are running FreeBSD, you need to edit the file **/etc/mail/mailer.conf** instead of setting up a symbolic link as just described. A typical example of the contents of this file for running Exim is as follows:

```
sendmail            /usr/exim/bin/exim
send-mail           /usr/exim/bin/exim
mailq               /usr/exim/bin/exim -bp
newaliases          /usr/bin/true
```

Once you have set up the symbolic link, or edited **/etc/mail/mailer.conf**, your Exim installation is 'live'. Check it by sending a message from your favourite user agent.

You should consider what to tell your users about the change of MTA. Exim may have different capabilities to what was previously running, and there are various operational differences such as the text of messages produced by command line options and in bounce messages. If you allow your users to make use of Exim's filtering capabilities, you should make the document entitled *Exim's interface to mail filtering* available to them.

4.17 Upgrading Exim

If you are already running Exim on your host, building and installing a new version automatically makes it available to MUAs, or any other programs that call the MTA directly. However, if you are running an Exim daemon, you do need to send it a HUP signal, to make it re-exec itself, and thereby pick up the new binary. You do not need to stop processing mail in order to install a new version of Exim.

4.18 Stopping the Exim daemon on Solaris

The standard command for stopping the mailer daemon on Solaris 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 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.

Note, however, that stopping the daemon does not ‘stop Exim’. Messages can still be received from local processes, and if automatic delivery is configured (the normal case), deliveries will still occur.

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. The **-bp** option requests a listing of the contents of the mail queue on the standard output. This feature is for compatibility with some systems that contain a command of that name in one of the standard libraries, symbolically linked to **/usr/sbin/sendmail** or **/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*. In the description below, the phrases 'Exim user' and 'Exim group' mean the user and group defined by EXIM_USER and EXIM_GROUP in **Local/Makefile** or set by the **exim_user** and **exim_group** options. These do not necessarily have to use the name 'exim'.

- The trusted users are root, the Exim user, any user listed in the **trusted_users** configuration option, and any user whose current group or any supplementary group is one of those listed in the **trusted_groups** configuration option. Note that the Exim group is not automatically trusted.

  Trusted users are always 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). See the **untrusted_set_sender** option for a way of permitting non-trusted users to set envelope senders. For a trusted user, there is never any check on the contents of the *From:* header line, and a *Sender:* line is never added. Furthermore, any existing *Sender:* line in incoming local (non-TCP/IP) messages is not removed.

  Trusted users may also specify a host name, host address, interface address, protocol name, ident value, and authentication data when submitting a message locally. Thus, they are able to insert messages into Exim's queue locally that have the characteristics of messages received from a remote host. Untrusted users may in some circumstances use **-f**, but can never set the other values that are available to trusted users.

- The admin users are root, the Exim user, and any user that is a member of the Exim group or of any group listed in the **admin_groups** configuration option. The current group does not have to be one of these groups.

Admin users are permitted to list the queue, and to carry out certain operations on messages, 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, and full debugging output.

By default, the use of the **-M**, **-q**, **-R**, and **-S** 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.

Warning: If you configure your system so that admin users are able to edit Exim's configuration file, you are giving those users an easy way of getting root. There is further discussion of this issue at the start of chapter 6.

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.

--help This option causes Exim to output a few sentences stating what it is. The same output is generated if the Exim binary is called with no options and no arguments.

-B<*type*>

This is a Sendmail option for selecting 7 or 8 bit processing. Exim is 8-bit clean; it ignores this option.

-bd This option runs Exim as a daemon, awaiting incoming SMTP connections. Usually the **-bd** option is combined with the **-q**<*time*> option, to specify that the daemon should also initiate periodic queue runs.

The **-bd** option can be used only by an admin user. If either of the **-d** (debugging) or **-v** (verifying) options are set, the daemon does not disconnect from the controlling terminal. When running this way, it can be stopped by pressing ctrl-C.

By default, Exim listens for incoming connections to the standard SMTP port on all the host's interfaces. The port can be varied by means of the **daemon_smtp_port** option. The daemon can also be restricted to specific interfaces by setting the **local_interfaces** option in the configuration file. This option is also able to specify a different port for each interface it lists, making it possible to listen on multiple ports. The **-oX** command line option can be used to override **local_interfaces**.

When a listening daemon is started without the use of **-oX** (that is, without overriding the normal configuration), it writes its process id to a file called **exim-daemon.pid** in Exim's spool directory. This location can be overridden by setting PID_FILE_PATH in **Local/Makefile**. The file is written while Exim is still running as root.

When **-oX** is used on the command line to start a listening daemon, the process id is not written to the normal pid file path. However, **-oP** can be used to specify a path on the command line if a pid file is required.

The SIGHUP signal can be used to cause the daemon to re-exec itself. This should be done whenever Exim's configuration file, or any file that is incorporated into it by means of the **.include** facility, is changed, and also whenever a new version of Exim is installed. It is not necessary to do this when other files that are referenced from the configuration (for example, alias files) are changed, because these are reread each time they are used.

-bdf This option has the same effect as **-bd** except that it never disconnects from the controlling terminal, even when no debugging is specified.

-be Run Exim in expansion testing mode. Exim discards its root privilege, to prevent ordinary users from using this mode to read otherwise inaccessible files. If no arguments are given, Exim runs interactively, prompting for lines of data. Long expressions can be split over several lines by using backslash continuations. As in Exim's run time configuration, whitespace at the start of continuation lines is ignored.

Each argument or data line is passed through the string expansion mechanism, and the result is output. Variable values from the configuration file (for example, **$qualify_domain**) are available, but no message-specific values (such as **$domain**) are set, because no message is being processed.

-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>*

This option runs 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
```

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 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 to the filter 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. The IP address may include a port number at the end, after a full stop. For example:

```
exim -bh 10.9.8.7.1234
exim -bh fe80::a00:20ff:fe86:a061.5678
```

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.

Warning: You cannot test features of the configuration that rely on ident (RFC 1413) callouts, because these are not done when testing using **-bh**.

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 and port if this is important.

The *exim_checkaccess* utility is a 'packaged' version of **-bh** whose output just states whether a given recipient address from a given host is acceptable or not. See section 45.7.

-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** with the **-bi** option 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, calling Exim with **-bi** is a no-op.

-bm This option runs an Exim receiving process that accepts an incoming, locally-generated message on the current input. The recipients are given as the command arguments (except when **-t** is also present – see below). Each argument can be a comma-separated list of RFC 2822 addresses. This is the default option for selecting the overall action of an Exim call; it is assumed if no other conflicting option is present.

If any addresses in the message are unqualified (have no domain), they are qualified by the values of the **qualify_domain** or **qualify_recipient** options, as appropriate. The **-bnq** option (see below) provides a way of suppressing this for special cases.

Policy checks on the contents of local messages can be enforced by means of the non-SMTP ACL. See chapter 37 for details. The return code is zero if the message is successfully accepted. Otherwise, the action is controlled by the **-oe***x* option setting – see below.

The format of the message must be as defined in RFC 2822, 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.

-bnq By default, Exim automatically qualifies unqualified addresses (those without domains) that appear in messages that are submitted locally (that is, not over TCP/IP). This qualification applies both to addresses in envelopes, and addresses in header lines. Sender addresses are qualified using **qualify_domain**, and recipient addresses using **qualify_recipient** (which defaults to the value of **qualify_domain**).

Sometimes, qualification is not wanted. For example, if **-bS** (batch SMTP) is being used to re-submit messages that originally came from remote hosts after content scanning, you probably do not want to qualify unqualified addresses in header lines. (Such lines will be present only if you have not enabled a header syntax check in the appropriate ACL.)

The **-bnq** option suppresses all qualification of unqualified addresses in messages that originate on the local host. When this is used, unqualified addresses in the envelope provoke errors (causing message rejection) and unqualified addresses in header lines are left alone.

-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 hold_domains
```

However, any option setting that is preceded by the word 'hide' in the configuration file is not shown in full, except to an admin user. For other users, the output is as in this example:

```
mysql_servers = <value not displayable>
```

If **configure_file** is given as an argument, the name of the run time configuration file is output. If a list of configuration files was supplied, the value that is output here is the name of the file that was actually used.

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 the pid file is written directly into the spool directory.

If **-bP** is followed by a name preceded by +, for example,

```
exim -bP +local_domains
```

it searches for a matching named list of any type (domain, host, address, or local part) and outputs what it finds.

If one of the words **router**, **transport**, or **authenticator** 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 **router_list**, **transport_list**, or **authenticator_list**, and a complete list of all drivers with their option settings can be obtained by using **routers**, **transports**, or **authenticators**.

-bp This option requests a listing of the contents of the mail queue on the standard output. If the **-bp** option is followed by a list of message ids, just those messages are listed. By default, this option can be used only by an admin user. However, the **queue_list_requires_admin** option can be set false to allow any user to see the queue.

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

```
25m  2.9K 0t5C6f-0000c8-00 <alice@wonderland.fict.example>
           red.king@looking-glass.fict.example
           <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. For bounce messages, the sender address is empty, and appears as '<>'. If the message was submitted locally by an untrusted user who overrode the default sender address, the user's login name is shown in parentheses before the sender address. 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'.

-bpc This option counts the number of messages on the queue, and writes the total to the standard output. It is restricted to admin users, unless **queue_list_requires_admin** is set false.

-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 router with the **one_time** option set.

-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.example
Retry rule: *.comp.mus.example  F,2h,15m; F,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 30 for further details.

-bS This option is used for batched SMTP input, which is an alternative interface for non-interactive local message submission. A number of messages can be submitted in a single run. However, despite its name, this is not really SMTP input. Exim reads each message's envelope from SMTP commands on the standard input, but generates no responses. If the caller is trusted, or **untrusted_set_sender** is set, the senders in the SMTP MAIL commands are believed; otherwise the sender is always the caller of Exim.

The message itself is read from the standard input, in SMTP format (leading dots doubled), terminated by a line containing just a single dot. An error is provoked if the terminating dot is missing. A further message may then follow.

As for other local message submissions, the contents of incoming batch SMTP messages can be checked using the non-SMTP ACL (see chapter 37). Unqualified addresses are automatically qualified using **qualify_domain** and **qualify_recipient**, as appropriate, unless the **-bnq** option is used.

Some other SMTP commands are recognized in the input. HELO and EHLO act as RSET; VRFY, EXPN, ETRN, and HELP act as NOOP; QUIT quits, ignoring the rest of the standard input.

If any error is encountered, reports are written to the standard output and error streams, and Exim gives up immediately. 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 42.12.

-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. SMTP policy controls, as defined in ACLs (see chapter 37) are applied.

Some user agents use this interface as a way of passing locally-generated messages to the MTA. In this usage, if the caller of Exim is trusted, or **untrusted_set_sender** is set, 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. Unqualified addresses are automatically qualified using **qualify_domain** and **qualify_recipient**, as appropriate, unless the **-bnq** option is used.

The **-bs** option is also used to run Exim from *inetd*, as an alternative to using a listening daemon. Exim can distinguish the two cases by checking whether the standard input is a TCP/IP socket. When Exim is called from *inetd*, the source of the mail is assumed to be remote, and the comments above concerning senders and qualification do not apply. In this situation, Exim behaves in exactly the same way as it does when receiving a message via the listening daemon.

-bt This option runs Exim in address testing mode, in which each argument is taken as an address to be tested for deliverability. The results are written to the standard output. If a test fails, and the caller is not an admin user, no details of the failure are output, because these might contain sensitive information such as usernames and passwords for database lookups.

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 (compare the **-bv** option). It is passed to the routers and the result is written to the standard output. However, any router that has **no_address_test** set is bypassed. This can make **-bt** easier to use for genuine routing tests if your first router passes everything to a scanner program.

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.

Warning: **-bt** can only do relatively simple testing. If any of the routers in the configuration makes any tests on the sender address of a message, you can 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. However, if you have set up (for example) routers whose behaviour depends on the contents of an incoming message, you cannot test those conditions using **-bt**. The **-N** option provides a possible way of doing such tests.

-bV This option causes Exim to write the current version number, compilation number, and compilation date of the *exim* binary to the standard output. It also lists the DBM library this is being used, the optional modules (such as specific lookup types), the drivers that are included in the binary, and the name of the run time configuration file that is in use.

-bv This option runs Exim in address verification mode, in which each argument is taken as an address to be verified. During normal operation, verification happens mostly as a consequence processing a **verify** condition in an ACL (see chapter 37). If you want to test an entire ACL, see the **-bh** option.

If verification fails, and the caller is not an admin user, no details of the failure are output, because these might contain sensitive information such as usernames and passwords for database lookups.

If no arguments are given, Exim runs in an interactive manner, prompting with a right angle bracket for addresses to be verified. Verification differs from address testing (the **-bt** option) in that routers that have **no_verify** set are skipped, and if the address is accepted by a router that has **fail_verify** set, verification fails. The address is verified as a recipient if **-bv** is used; to test verification for a sender address, **-bvs** should be used.

If the **-v** 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 address redirection, all the generated addresses are also considered. Without **-v**, generating more than one address by redirection causes verification to end sucessfully.

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 routers in the configuration makes any tests on the sender address of a message, 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 *<filelist>*

This option causes Exim to find the run time configuration file from the given list instead of from the list specified by the CONFIGURE_FILE compile-time setting. Usually, the list will consist of just a single file name, but it can be a colon-separated list of names. In this case, the first file that exists is used. Failure to open an existing file stops Exim from proceeding any further along the list, and an error is generated.

When this option is used by a caller other than root or the Exim user, and the list is different from the compiled-in list, Exim gives up its root privilege immediately, and runs with the real and effective uid and gid set to those of the caller.

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 is an exotic configuration that does not require privilege. No check is made on the owner or group of the files specified by this option.

-D<*macro*>=<*value*>

This option can be used to override macro definitions in the configuration file (see section 6.4). However, like **-C**, if it is used by an unprivileged caller, it causes Exim to give up its root privilege. The entire option (including equals sign if present) must all be within one command line item. **-D** can be used to set the value of a macro to the empty string, in which case the equals sign is optional. These two commands are synonymous:

```
exim -DABC  ...
exim -DABC= ...
```

To include spaces in a macro definition item, quotes must be used. If you use quotes, spaces are permitted around the macro name and the equals sign. For example:

```
exim '-D ABC = something' ...
```

-D may be repeated up to 10 times on a command line.

-d<*debug options*>

This option causes debugging information to be written to the standard error stream. It is restricted to admin users because debugging output may show database queries that contain password information. Also, the details of users' filter files should be protected. When **-d** is used, **-v** is assumed. If **-d** is given on its own, a lot of standard debugging data is output. This can be reduced, or increased to include some more rarely needed information, by following **-d** with a string made up of names preceded by plus or minus characters. These add or remove sets of debugging data, respectively. For example, **-d+filter** adds filter debugging, whereas **-d-all+filter** selects only filter debugging. The available debugging categories are:

`acl`	ACL interpretation
`auth`	authenticators
`deliver`	general delivery logic
`dns`	DNS lookups (see also resolver)
`dnsbl`	DNS black list (aka RBL) code
`exec`	arguments for **execv()** calls
`expand`	detailed debugging for string expansions
`filter`	filter handling
`hints_lookup`	hints data lookups
`host_lookup`	all types of name-to-IP address handling
`ident`	ident lookup
`interface`	lists of local interfaces

`lists`	matching things in lists
`load`	system load checks
`local_scan`	can be used by *local_scan()* (see chapter 38)
`lookup`	general lookup code and all lookups
`memory`	memory handling
`pid`	add pid to debug output lines
`process_info`	setting info for the process log
`queue_run`	queue runs
`receive`	general message reception logic
`resolver`	turn on the DNS resolver's debugging output
`retry`	retry handling
`rewrite`	address rewriting
`route`	address routing
`timestamp`	add timestamp to debug output lines
`tls`	TLS logic
`transport`	transports
`uid`	changes of uid/gid and looking up uid/gid
`verify`	address verification logic
`all`	all of the above, and also **-v**

Unfortunately, debugging output from the DNS resolver is written to stdout rather than stderr.

The default (**-d** with no argument) omits `expand`, `filter`, `interface`, `load`, `memory`, `pid`, `resolver`, and `timestamp`. However, the `pid` selector is forced when debugging is turned on for a daemon, which then passes it on to any re-executed Exims. Exim also automatically adds the pid to debug lines when several remote deliveries are run in parallel.

The `timestamp` selector causes the current time to be inserted at the start of all debug output lines. This can be useful when trying to track down delays in processing.

If the **debug_print** option is set in any driver, it produces output whenever any debugging is selected, or if **-v** is used.

-dropcr

The RFCs that define Internet mail apply only to messages in transit between hosts. They specify that lines of text should be terminated by the two-character sequence CR, LF. When a message is within a host system, the software that processes it may use any method it likes for terminating lines. The natural assumption is to use the host's normal convention. Most software on Unix-like systems uses a single LF character, which is the Unix standard.

When a non-SMTP message is passed to Exim via its command line, LF termination is assumed. Any CR characters in the message, wherever they appear, are treated as data characters.

Unfortunately, not all software writers take the same view. At least one MUA (dtmail) terminates each line with CR, LF, and the Cyrus message store behaves in the same way. There is also some UUCP software which does this. To support these callers, Exim has the **-dropcr** option, which causes it to discard a CR character if it immediately precedes an LF. Any other CR characters are treated as data. For example, a sequence such as CR, CR, LF is treated as one data CR, followed by the end of the line. The **drop_cr** configuration file option can be used to force **-dropcr** for all non-SMTP input.

-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 postmaster, 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>*
This option sets 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 data 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>*
This option sets the address of the envelope sender of a locally-generated message (also known as the return path). The option can normally be used only by a trusted user, but **untrusted_set_sender** can be set to allow untrusted users to use it. In the absence of **-f**, or if the caller is not allowed to use it, the sender of a local message is set to the caller's login name at the default qualify domain.

There is one exception to the restriction on the use of **-f**: an empty sender can be specified by any user, to create a message that can never provoke a bounce. An empty sender can be specified either as an empty string, or as a pair of angle brackets with nothing between them, as in these examples of shell commands:

```
exim -f '<>' user@domain
exim -f "" user@domain
```

In addition, the use of **-f** is not restricted when testing a filter file with **-bf** or when testing or verifying addresses using the **-bt** or **-bv** options.

Allowing untrusted users to change the sender address does not of itself make it possible to send anonymous mail. Exim still checks that the *From:* header refers to the local user, and if it does not, it adds a *Sender:* header, though this can be overridden by setting **no_local_from_check**.

White space between **-f** and the *<address>* is optional (that is, they can be given as two arguments or one combined argument). The sender of a locally-generated message can also be set (when permitted) by an initial 'From ' line in the message – see the description of **-bm** above – but if **-f** is also present, it overrides 'From'.

-G This is a Sendmail option which is ignored by Exim.

-h *<number>*
This option is accepted for compatibility with Sendmail, but 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 *<message id> <message id> ...*
This option requests Exim to run a delivery attempt on each message in turn. If any of the messages are frozen, they are automatically thawed before the delivery attempt. The settings of **queue_domains**, **queue_smtp_domains**, and **hold_domains** are ignored. 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**, **-R**, and **-S** options).

-Mar *<message id> <address> <address> ...*
This option requests Exim to add the addresses to the list of recipients of the message ('ar' for 'add recipients'). The first argument must be a message id, and the remaining ones must be email addresses. However, if the message is active (in the middle of a delivery attempt), it 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 connection, which is passed as the standard input. 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.

-MCA This option is not intended for use by external callers. It is used internally by Exim in conjunction with the **-MC** option. It signifies that the connection to the remote host has been authenticated.

-MCP This option is not intended for use by external callers. It is used internally by Exim in conjunction with the **-MC** option. It signifies that the server to which Exim is connected supports pipelining.

-MCQ *<process id> <pipe fd>*
This option is not intended for use by external callers. It is used internally by Exim in conjunction with the **-MC** option when the original delivery was started by a queue runner. It passes on the process id of the queue runner, together with the file descriptor number of an open pipe. Closure of the pipe signals the final completion of the sequence of processes that are passing messages through the same SMTP connection.

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

-MCT This option is not intended for use by external callers. It is used internally by Exim in conjunction with the **-MC** option, and passes on the fact that the host to which Exim is connected supports TLS encryption.

-Mc *<message id> <message id> ...*
This option requests Exim to run 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. It is provided mainly for internal use by Exim when it needs to re-invoke itself in order to regain root privilege for a delivery (see chapter 47). However, **-Mc** can be useful when testing, in order to run a delivery that respects retry times and other options such as **hold_domains** that are overridden when **-M** is used. Such a delivery does not count as a queue run. If you want to run a specific delivery as if in a queue run, you should use **-q** with a message id argument. A distinction between queue run deliveries and other deliveries is made in one or two places.

-Mes *<message id> <address>*
This option requests Exim to change the sender address in the message to the given address, which must be a fully qualified address or '<>' ('es' for 'edit sender'). There must be exactly two arguments. The first argument must be a message id, and the second one an email address. 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 *<message id> <message id> ...*
This option requests Exim to mark each listed message as '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 *<message id> <message id> ...*
This option requests Exim to give up trying to deliver the listed messages, including any that are frozen. However, if any of the messages are active, their status is not altered. For non-bounce messages, a delivery error message is sent to the sender, containing the text 'cancelled by administrator'. Bounce messages are just discarded. This option can be used only by an admin user.

-Mmad *<message id> <message id> ...*
This option requests Exim to mark all the recipient addresses in the messages as already delivered (‘mad’ for ‘mark all delivered’). However, if any 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> ...*
This option requests Exim to mark the given addresses as already delivered (‘md’ for ‘mark delivered’). The first argument must be a message id, and the remaining ones must be email addresses. These are matched to recipient addresses in the message in a case-sensitive manner. 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 *<message id> <message id> ...*
This option requests Exim to remove the given messages from the queue. No bounce messages are sent; each message is simply 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 *<message id> <message id> ...*
This option requests Exim to ‘thaw’ any of the listed messages that are ‘frozen’, 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>*
This option causes the contents of the message body (-D) spool file to be written to the standard output. This option can be used only by an admin user.

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

-Mvl *<message id>*
This option causes the contents of the message log spool file to be 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 **-v**. 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 ‘=>’.

Because **-N** discards any message to which it applies, only root or the Exim user are allowed to use it 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. Although transportation never fails when **-N** is set, an address may be deferred because of a configuration problem on a transport, or a routing problem. Once **-N** has been used for a delivery attempt, it sticks to the message, and applies to any subsequent delivery attempts that may happen for that message.

-n This option is interpreted by Sendmail to mean ‘no aliasing’. It is ignored by Exim.

-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 messages that can be delivered down one SMTP connection, overriding the value set in any **smtp** transport. If *<n>* is omitted, the limit is set to 1.

-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 delivery process for each message received, but does not wait for the delivery process to complete. This is the default action if none of the **-od** options are present. It overrides a setting of **queue_only** in the configuration file.

-odf This option requests 'foreground' (synchronous) delivery when Exim has accepted a locally-generated message. (For the daemon it is exactly the same as **-odb**.) A delivery process is automatically started to deliver the message, and Exim 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 process for each message received. Messages are placed on the queue, and remain there until a subsequent queue runner process encounters them. The **queue_only** configuration option has the same effect. This option overrides any setting of **queue_smtp_domains** or **-odqs**.

-odqs This option is a hybrid between **-odb** and **-odq**. A background delivery process is started for each incoming message, the addresses are routed, and local deliveries are done in the normal way. However, if any SMTP deliveries are required, they are not done at this time, so the message remains on the queue until a subsequent queue runner process encounters it. Because routing was done, Exim knows which messages are waiting for which hosts, and so a number of messages for the same host can be sent in a single SMTP connection. The **queue_smtp_domains** 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 this error message is successfully sent, the Exim receiving process exits with a return code of zero. If not, the return code is 2 if the problem is that the original message has no recipients, or 1 any other error. This is the default **-oe***x* option 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 **-oe***x* 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). The return code is 1 for all errors.

-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*.

-oitrue This option is treated as synonymous with **-oi**.

-oMa *<host address>*
A number of options starting with **-oM** can be used to set values associated with remote hosts on locally-submitted messages (that is, messages not received over TCP/IP). These options can be used by any caller in conjunction with the **-bh**, **-bf**, **-bF**, **-bt**, or **-bv** testing options. In other circumstances, they are ignored unless the caller is trusted.

The **-oMa** option sets the sender host address. This may include a port number at the end, after a full stop (period). For example:

```
exim -bs -oMa 10.9.8.7.1234
```

An alternative syntax is to enclose the IP address in square brackets, followed by a colon and the port number:

```
exim -bs -oMa [10.9.8.7]:1234
```

-oMaa *<name>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMaa** option sets the value of **$sender_host_authenticated** (the authenticator name). See chapter 32 for a discussion of SMTP authentication.

-oMai *<string>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMai** option sets the authenticated id value. It overrides the default value (the caller's login id) for messages from local sources. See chapter 32 for a discussion of authenticated ids.

-oMas *<address>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMas** option sets the authenticated sender value. It overrides the sender address that is created from the caller's login id for messages from local sources. See chapter 32 for a discussion of authenticated senders.

-oMi *<interface address>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMi** option sets the IP interface address value. A port number may be included, using the same syntax as for **-oMa**.

-oMr *<protocol name>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMr** option sets the received protocol value. However, this applies only when **-bs** is not used. For interactive SMTP input, the protocol is determined by whether EHLO or HELO is used, and is always either 'local-esmtp' or 'local-smtp'. For **-bS** (batch SMTP) however, the protocol can be set by **-oMr**.

-oMs *<host name>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMs** option sets the sender host name.

-oMt *<ident string>*
See **-oMa** above for general remarks about the **-oM** options. The **-oMt** option sets the sender ident value. The default setting for local callers is the login id of the calling process.

-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.

-oo This option is ignored. In Sendmail it specifies 'old style headers', whatever that means.

-oP *<path>*
This option is useful only in conjunction with **-bd** or **-q** with a time value. The option specifies the file to which the process id of the daemon is written. When **-oX** is used with **-bd**, or when **-q** with a time is used without **-bd**, this is the only way of causing Exim to write a pid file, because in those cases, the normal pid file is not used.

-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 by the **receive_timeout** option. The format used for specifying times is described in section 6.11.

-os *<time>*
This option sets a timeout value for incoming SMTP messages. The timeout applies to each SMTP command and block of data. The value can also be set by the **smtp_receive_timeout** option; it defaults to 5 minutes. The format used for specifying times is described in section 6.11.

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

-oX *<number or string>*

This option is relevant only when the **-bd** option is also given. If it is followed by a single number, it specifies the default TCP port for the listening daemon, overriding **daemon_smtp_port**. If **local_interfaces** is set, and specifies ports as well as IP addresses, **-oX** followed by a single number has no effect, because it changes only the default port.

An alternate form for **-oX** is to follow it with a list of interfaces (and optionally ports) on which the daemon is to listen. In this case, the data is in the same format as the value of **local_interfaces**, and it overrides that option.

-pd This option applies when an embedded Perl interpreter is linked with Exim (see chapter 12). 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 12). 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 This option is normally restricted to admin users. However, there is a configuration option called **prod_requires_admin** which can be set false to relax this restriction (and also the same requirement for the **-M**, **-R**, and **-S** options).

The **-q** option starts one queue runner process. This scans the queue of waiting messages, and runs a delivery process for each one in turn. It waits for each delivery process to finish before starting the next one. A delivery process may not actually do any deliveries if the retry times for the addresses have not been reached. Use **-qf** (see below) if you want to override this. 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 queue runner process terminates. In other words, a single pass is made over the waiting mail, one message at a time. 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.

It is possible to cause the messages to be processed in lexical message id order, which is essentially the order in which they arrived, by setting the **queue_run_in_order** option, but this is not recommended for normal use.

-q<*qflags*>

The **-q** option may be followed by one or more flag letters that change its behaviour. They are all optional, but if more than one is present, they must appear in the correct order. Each flag is described in a separate item below.

-qq... An option starting with **-qq** requests a two-stage queue run. In the first stage, the queue is scanned as if the **queue_smtp_domains** option matched every domain. Addresses are routed, local deliveries happen, but no remote transports are run. The hints 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, with routing and delivery taking place as normal. Messages that 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.

-q[q]i...

If the *i* flag is present, the queue runner runs delivery processes only for those messages that haven't previously been tried. (*i* stands for 'initial delivery'.) This can be helpful if you are putting messages on the queue using **-odq** and want a queue runner just to process the new messages.

-q[q][i]f...

If one *f* flag is present, a delivery attempt is forced for each non-frozen message, whereas without **f** only those non-frozen addresses that have passed their retry times are tried.

-q[q][i]ff...

If *ff* is present, a delivery attempt is forced for every message, whether frozen or not.

-q[q][i][f[f]]l

The *l* (the letter 'ell') flag specifies that only local deliveries are to be done. If a message requires any remote deliveries, it remains on the queue for later delivery.

-q<*qflags*> <*start id*> <*end id*>

When scanning the queue, Exim can be made to skip over messages whose ids are lexically less than a given value by following the **-q** option with a starting message id. For example:

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

Messages that arrived earlier than `0t5C6f-0000c8-00` are not inspected. If a second message id is given, messages whose ids are lexically greater than it are also skipped. If the same id is given twice, for example,

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

just one delivery process is started, for that message. This differs from **-M** in that retry data is respected, and it also differs from **-Mc** in that it counts as a delivery from a queue run. Note that the selection mechanism does not affect the order in which the messages are scanned. There are also other ways of selecting specific sets of messages for delivery in a queue run – see **-R** and **-S**.

-q<*qflags*><*time*>

When a time value is present, the **-q** option causes Exim to run as a daemon, starting a queue runner process at intervals specified by the given time value (whose format is described in section 6.11). 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

```
/usr/exim/bin/exim -bd -q30m
```

Such a daemon listens for incoming SMTP calls, and also starts a queue runner process every 30 minutes.

When a daemon is started by **-q** with a time value, but without **-bd**, no pid file is written unless one is explicitly requested by the **-oP** option.

-qR<*rsflags*> <*string*>

This option is synonymous with **-R**. It is provided for Sendmail compatibility.

-qS<*rsflags*> <*string*>

This option is synonymous with **-S**.

-R<*rsflags*> <*string*>

The <*rsflags*> may be empty, in which case the white space before the string is optional, unless the string is *f*, *ff*, *r*, *rf*, or *rff*, which are the possible values for <*rsflags*>. White space is required if <*rsflags*> is not empty.

This option is similar to **-q** with no time value, that is, it causes Exim to perform a single queue run, except that, when scanning the messages on the queue, Exim processes only those that have at least one undelivered recipient address containing the given string, which is checked in a case-independent way. If the <*rsflags*> start with *r*, <*string*> is interpreted as a regular expression; otherwise it is a literal string.

Once a message is selected, all its addresses are processed. For the first selected message, Exim overrides any retry information and forces a delivery attempt for each undelivered address. This means that if delivery of any address in the first message is successful, any

existing retry information is deleted, and so delivery attempts for that address in subsequently selected messages (which are processed without forcing) will run. However, if delivery of any address does not succeed, the retry information is updated, and in subsequently selected messages, the failing address will be skipped.

If the *<rsflags>* contain *f* or *ff*, the delivery forcing applies to all selected messages, not just the first; frozen messages are included when *ff* is present.

The **-R** option 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 accepted by its ACL (see chapter 37), its default effect is to run Exim with the **-R** option, but it can be configured to run an arbitrary command instead.

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

-S*<rsflags>* *<string>*

This option acts like **-R** except that it checks the string against each message's sender instead of against the recipients. If **-R** is also set, both conditions must be met for a message to be selected. If either of the options has *f* or *ff* in its flags, the associated action is taken.

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

If the command has 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 several versions of Sendmail, as described in man pages on a number of operating systems (e.g. Solaris 8, IRIX 6.5, HP-UX 11). However, some versions of Sendmail *add* argument addresses to those obtained from the headers, and the O'Reilly Sendmail book documents it that way. Exim can be made to add argument addresses instead of subtracting them by setting the option **extract_addresses_remove_arguments** false.

If a *Bcc:* header line is present, it is removed from the message unless there is no *To:* or *Cc:*, in which case a *Bcc:* line with no data is created. This is necessary for conformity with the original RFC 822 standard; the requirement has been removed in RFC 2822, but that is still very new.

If there are any **Resent-** header lines in the message, Exim extracts recipients from all *Resent-To:*, *Resent-Cc:*, and *Resent-Bcc:* header lines instead of from *To:*, *Cc:*, and *Bcc:*. This is for compatibility with Sendmail and other MTAs. (Prior to release 4.20, Exim gave an error if **-t** was used in conjunction with **Resent-** header lines.)

RFC 2822 talks about different sets of **Resent-** header lines (for when a message is resent several times). The RFC also specifies that they should be added at the front of the message, and separated by *Received:* lines. It is not at all clear how **-t** should operate in the present of multiple sets. In practice, it seems that MUAs do not follow the RFC. The **Resent-** lines are often added at the end of the header, and if a message is resent more than once, it is common for the original set of **Resent-** headers to be renamed as **X-Resent-** when a new set is added. This removes any possible ambiguity.

-tls-on-connect

This option is available when Exim is compiled with TLS support. It makes it possible to support legacy clients that do not support the STARTTLS command, but instead expect to start up a TLS session as soon as a connection to the server is established. These clients use a special port (usually called the 'ssmtp' port) instead of the normal SMTP port 25. The **-tls-on-connect** option can be used to run Exim in this way from *inetd*, and it can also be used to run a special daemon that operates in this manner (use **-oX** to specify the port). However, although it is possible to run one daemon that listens on several ports, it is not possible to have some of them operate one way and some the other. With only a few clients that need the

legacy support, a convenient approach is to use a daemon for normal SMTP (with or without STARTTLS) and *inetd* with **-tls-on-connect** for the legacy clients.

-U Sendmail uses this option for ‘initial message submission’, and its documentation states that in future releases, it may complain about syntactically invalid messages rather than fixing them when this flag is not set. Exim ignores this option.

-v This option causes Exim to write information to the standard error stream, describing what it is doing. In particular, it shows the log lines for receiving and delivering a message, and if an SMTP connection is made, the SMTP dialogue is shown. Some of the log lines shown may not actually be written to the log if the setting of **log_selector** discards them. Any relevant selectors are shown with each log line. If none are shown, the logging is unconditional.

-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. The Exim run time configuration file

Exim uses a single run time configuration file that is read whenever an Exim binary is executed. Note that in normal operation, this happens frequently, because Exim is designed to operate in a distributed manner, without central control.

The name of the configuration file is compiled into the binary for security reasons, and is specified by the CONFIGURE_FILE compilation option. In most configurations, this specifies a single file. However, it is permitted to give a colon-separated list of file names, in which case Exim uses the first existing file in the list.

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

Warning: In a conventional configuration, where the Exim binary is setuid to root, anybody who is able to edit the run time configuration file has an easy way to run commands as root. If you make your mail administrators members of the Exim group, but do not trust them with root, make sure that the run time configuration is not group writeable.

A default configuration file, which will work correctly in simple situations, is provided in the file **src/configure.default**. If CONFIGURE_FILE defines just one file name, the installation process copies the default configuration to a new file of that name if it did not previously exist. If CONFIGURE_FILE is a list, no default is automatically installed. Chapter 7 is a 'walk-through' discussion of the default configuration.

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

6.1 Using a different configuration file

A one-off alternate configuration can be specified by the **-C** command line option, which may specify a single file or a list of files. However, when **-C** is used, Exim gives up its root privilege, unless called by root or the Exim user (or unless the argument for **-C** is identical to the built-in value from CONFIGURE_FILE). **-C** 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**.

One-off changes to a configuration can be specified by the **-D** command line option, which defines and overrides values for macros used inside the configuration file. However, like **-C**, the use of this option by a non-privileged user causes Exim to discard its root privilege.

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**, 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. This processing occurs for each file name in the list given by CONFIGURE_FILE or **-C**.

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.

6.2 Configuration file format

Exim's configuration file is divided into a number of different parts. General option settings must always appear at the start of the file. The other parts are all optional, and may appear in any order. Each part other than the first is introduced by the word 'begin' followed by the name of the part. The optional parts are:

- *ACL*: Access control lists for controlling incoming SMTP mail.
- *authenticators*: Configuration settings for the authenticator drivers. These are concerned with the SMTP AUTH command (see chapter 32).
- *routers*: Configuration settings for the router drivers. Routers process addresses and determine how the message is to be delivered.
- *transports*: Configuration settings for the transport drivers. Transports define mechanisms for copying messages to destinations.
- *retry*: Retry rules, for use when a message cannot be immediately delivered.
- *rewrite*: Global address rewriting rules, for use when a message arrives and when new addresses are generated during delivery.
- *local_scan*: Private options for the *local_scan()* function. If you want to use this feature, you must set

  ```
  LOCAL_SCAN_HAS_OPTIONS=yes
  ```

  in **Local/Makefile** before building Exim. Full details of the *local_scan()* facility are given in chapter 38.

Blank lines in the file, and lines starting with a # character (ignoring leading white space) are treated as comments and are ignored. **Note**: a # character other than at the beginning of a line is not treated specially, and does not introduce a comment.

Any non-comment line can be continued by ending it with a backslash. Trailing white space after the backslash is ignored, and leading white space at the start of continuation lines is also ignored. Comment lines may appear in the middle of a sequence of continuation lines.

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 ACLs, retry rules, and rewriting rules have their own syntax which is described in chapters 37, 31, and 30, respectively. The other parts of the configuration file have some syntactic items in common, and these are described below, from section 6.6 onwards. Before that, the inclusion, macro, and conditional facilities are described.

6.3 File inclusions in the configuration file

You can include other files inside Exim's run time configuration file by using this syntax:

```
.include <file name>
```

on a line by itself. Double quotes round the file name are optional. Includes may be nested to any depth, but remember that Exim reads its configuration file often, so it is a good idea to keep them to a minimum.

If you change the contents of an included file, you must HUP the daemon, because an included file is read only when the configuration itself is read.

The processing of inclusions happens early, at a physical line level, so, like comment lines, an inclusion can be used in the middle of an option setting, for example:

```
hosts_lookup = a.b.c \
               .include /some/file
```

Include processing happens after macro processing (see below). Its effect is to process the lines of the file as if they occurred inline where the inclusion appears.

6.4 Macros in the configuration file

If a line in the main part of the configuration (that is, before the first 'begin' line) begins with an upper case letter, it is taken as a macro definition, and must be 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, including any continuations, is the replacement text, and has leading and trailing white space removed. Quotes are not removed. The 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 (and any included files) 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 is scanned 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 else>>
```

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

Macro expansion is applied to individual lines from the file, before checking for line continuation or file inclusion (see below). If a line consists solely of a macro name, and the expansion of the macro is empty, the line is ignored. A macro at the start of a line may turn the line into a comment line or a `.include` line.

As an example of macro usage, consider a configuration where aliases are looked up in a MySQL database. It helps to keep the file less cluttered if long strings such as SQL statements are defined separately as macros, for example:

```
ALIAS_QUERY = select mailbox from user where \
              login=${quote_mysql:$local_part};
```

This can then be used in a **redirect** router setting like this:

```
data = ${lookup mysql{ALIAS_QUERY}}
```

In earlier versions of Exim macros were sometimes used for domain, host, or address lists. In Exim 4 these are handled better by named lists – see section 10.5.

Macros in the configuration file can be overridden by the **-D** command line option, but Exim gives up its root privilege when **-D** is used, unless called by root or the Exim user.

6.5 Conditional skips in the configuration file

You can use the directives `.ifdef`, `.ifndef`, `.elifdef`, `.elifndef`, `.else`, and `.endif` to dynamically include or exclude portions of the configuration file. The processing happens whenever the file is read (that is, when an Exim binary starts to run).

The implementation is very simple. Instances of the first four directives must be followed by text that includes the names of one or macros. The condition that is tested is whether or not any macro substitution has taken place in the line. Thus:

```
.ifdef AAA
message_size_limit = 50M
.else
message_size_limit = 100M
.endif
```

sets a message size limit of 50M if the macro `AAA` is defined, and 100M otherwise. If there is more than one macro named on the line, the condition is true if any of them are defined. That is, it is an 'or' condition. To obtain an 'and' condition, you need to use nested `.ifdef`s.

Although you can use a macro expansion to generate one of these directives, it is not very useful, because the condition ‘there was a macro substitution in this line’ will always be true.

Text following `.else` and `.endif` is ignored, and can be used as comment to clarify complicated nestings.

6.6 Common option syntax

For the main set of options, driver options, and *local_scan()* 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:

```
qualify_domain = mydomain.example.com
```

Some option settings may contain sensitive data, for example, passwords for accessing databases. To stop non-admin users from using the **-bP** command line option to read these values, you can precede the option settings with the word ‘hide’. For example:

```
hide mysql_servers = localhost/users/admin/secret-password
```

For non-admin users, such options are displayed like this:

```
mysql_servers = <value not displayable>
```

If ‘hide’ is used on a driver option, it hides the value of that option on all instances of the same driver.

The following sections describe the syntax used for the different data types that are found in option settings.

6.7 Boolean options

Options whose type is given as boolean are on/off switches. There are two different ways of specifying such options: with and without a data value. If the option name is specified on its own without data, the switch is turned on; if it is preceded by ‘no_’ or ‘not_’ the switch is turned off. However, boolean options may optionally be followed by an equals sign and one of the words ‘true’, ‘false’, ‘yes’, or ‘no’, as an alternative syntax. For example, the following two settings have exactly the same effect:

```
queue_only
queue_only = true
```

The following two lines also have the same (opposite) effect:

```
no_queue_only
queue_only = false
```

You can use whichever syntax you prefer.

6.8 Integer values

If an integer 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 sometimes, but not always, printed using the letters K and M. The printing style is independent of the actual input format that was used.

6.9 Octal integer values

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.

6.10 Fixed point number values

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

6.11 Time interval values

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. Exim does not restrict the values; it is perfectly acceptable, for example, to specify ‘90m’ instead of ‘1h30m’.

6.12 String values

If a string data item does not start with a double-quote character, it is taken as consisting of the remainder of the line plus any continuation lines, starting at the first character after any leading white space, with trailing white space characters removed, and with no interpretation of the characters in the string. 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 settings are therefore equivalent:

```
trusted_users = uucp:mail

trusted_users = uucp:\
                # This comment line is ignored
                mail
```

If a string does start with a double-quote, it must end with a closing double-quote, and any backslash characters other than those used for line continuation are interpreted as escape characters, as follows:

`\\`	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, that character replaces the pair.

Quoting is necessary only if you want to make use of the backslash escapes to insert special characters, or if you need to specify a value with leading or trailing spaces. These cases are rare, so quoting is almost never needed in current versions of Exim. In versions of Exim before 3.14, quoting was required in order to continue lines, so you may come across older configuration files and examples that apparently quote unnecessarily.

6.13 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 (see chapter 11). The input syntax for such strings is as just described; in particular, the handling of backslashes in quoted strings is done as part of the input process, before expansion takes place. However, backslash is also an escape character for the expander, so any backslashes that are required for that reason must be doubled if they are within a quoted configuration string.

6.14 User and group names

User and group names are specified as strings, using the syntax described above, but the strings are interpreted specially. 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.

6.15 List construction

Some configuration settings accept a colon-separated list of items. 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 6.12 above is an example. If a colon is actually needed in an item in a list, it must be entered as two colons. Leading and trailing white space on each item in a 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, the list

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

contains two IP addresses, the IPv4 address 127.0.0.1 and the IPv6 address ::1. IPv6 addresses are going to become more and more common as the new protocol gets more widely deployed. Doubling their colons is an unwelcome chore, so a mechanism was introduced to allow the separator character to be changed. If a list begins with a left angle bracket, followed by any punctuation character, that character is used instead of colon as the list separator. For example, the list above can be rewritten to use a semicolon separator like this:

```
local_interfaces = <; 127.0.0.1 ; ::1
```

This facility applies to all lists, with the exception of the list in **log_file_path**. It is recommended that the use of non-colon separators be confined to circumstances where they really are needed.

6.16 Format of driver configurations

There are separate parts in the configuration for defining routers, transports, and authenticators. In each part, you are defining a number of driver instances, each with its own set of options. Each driver instance is defined by a sequence of lines like this:

```
<instance name>:
  <option>
  ...
  <option>
```

In the following example, the instance name is **localuser**, and it is followed by three options settings:

```
localuser:
  driver = accept
  check_local_user
  transport = local_delivery
```

For each driver instance, you specify which Exim code module it uses – by the setting of the **driver** option – and (optionally) some configuration settings. For example, in the case of transports, if you want a transport to deliver with SMTP you would use the **smtp** driver; if you want to deliver to a local file you would use the **appendfile** driver. Each of the drivers is described in detail in its own separate chapter later in this manual.

You can have several routers, transports, or authenticators that are based on the same underlying driver (each must have a different name).

The order in which routers are defined is important, because addresses are passed to individual routers one by one, in order. The order in which transports are defined does not matter at all. The order in which authenticators are defined is used only when Exim, as a client, is searching them to find one that matches an authentication mechanism offered by the server.

Within a driver instance definition, 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 routers, all transports or all

authenticators). The **driver** option is a generic option that must appear in every definition. The private options are special for each driver, and none need appear, because they all have default values.

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.

Driver instance names, which are used for reference in log entries and elsewhere, 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 module. For example, 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
  port = 1234
  command_timeout = 10s
```

The names **remote_smtp** and **special_smtp** would be used to reference these transport instances from routers, and these names would appear in log lines.

Comment lines may be present in the middle of driver specifications. The full list of option settings for any particular driver instance, including all the defaulted values, can be extracted by making use of the **-bP** command line option.

7. The default configuration file

The default configuration file supplied with Exim as **src/configure.default** is sufficient for a host with simple mail requirements. As an introduction to the way Exim is configured, this chapter 'walks through' the default configuration, giving brief explanations of the settings. Detailed descriptions of the options are given in subsequent chapters. The default configuration file itself contains extensive comments about ways you might want to modify the initial settings. However, note that there are many options that are not mentioned at all in the default configuration.

7.1 Main configuration settings

The main (global) configuration option settings must always come first in the file. The first thing you'll see in the file, after some initial comments, is the line

```
# primary_hostname =
```

This is a commented-out setting of the **primary_hostname** option. Exim needs to know the official, fully qualified name of your host, and this is where you can specify it. However, in most cases you do not need to set this option. When it is unset, Exim uses the *uname()* system function to obtain the host name.

The first three non-comment configuration lines are as follows:

```
domainlist local_domains = @
domainlist relay_to_domains =
hostlist   relay_from_hosts = 127.0.0.1
```

These are not, in fact, option settings. They are definitions of two named domain lists and one named host list. Exim allows you to give names to lists of domains, hosts, and email addresses, in order to make it easier to manage the configuration file (see section 10.5).

The first line defines a domain list called *local_domains*; this is used later in the configuration to identify domains that are to be delivered on the local host. There is just one item in this list, the string '@'. This is a special form of entry which means 'the name of the local host'. Thus, if the local host is called *a.host.example*, mail to *any.user@a.host.example* is expected to be delivered locally. Because the local host's name is referenced indirectly, the same configuration file can be used on different hosts.

The second line defines a domain list called *relay_to_domains*, but the list itself is empty. Later in the configuration we will come to the part that controls mail relaying through the local host; it allows relaying to any domains in this list. By default, therefore, no relaying on the basis of a mail domain is permitted.

The third line defines a host list called *relay_from_hosts*. This list is used later in the configuration to permit relaying from any host or IP address that matches the list. The default contains just the IP address of the IPv4 loopback interface, which means that processes on the local host are able to submit mail for relaying by sending it over TCP/IP to that interface. No other hosts are permitted to submit messages for relaying.

Just to be sure there's no misunderstanding: at this point in the configuration we aren't actually setting up any controls. We are just defining some domains and hosts that will be used in the controls that are specified later.

The next configuration line is a genuine option setting:

```
acl_smtp_rcpt = acl_check_rcpt
```

This option specifies an *Access Control List* (ACL) which is to be used during an incoming SMTP session for every recipient of a message (every RCPT command). The name of the list is *acl_check_rcpt*, and we will come to its definition below, in the ACL section of the configuration.

ACLs control which recipients are accepted for an incoming message – if a configuration does not provide an ACL to check recipients, no SMTP mail can be accepted.

Two commented-out options settings are next:

```
# qualify_domain =
# qualify_recipient =
```

The first of these specifies a domain that Exim uses when it constructs a complete email address from a local login name. This is often needed when Exim receives a message from a local process. If you do not set **qualify_domain**, the value of **primary_hostname** is used. If you set both of these options, you can have different qualification domains for sender and recipient addresses. If you set only the first one, its value is used in both cases.

The following line must be uncommented if you want Exim to recognize addresses of the form *user@[10.11.12.13]* that is, with a 'domain literal' (an IP address) instead of a named domain.

```
# allow_domain_literals
```

The RFCs still require this form, but in the modern Internet it makes little sense to permit mail to be sent to specific hosts by quoting their IP addresses. This ancient format has been used by those seeking to abuse hosts by using them for unwanted relaying.

The next configuration line is a kind of trigger guard:

```
never_users = root
```

It specifies that no delivery must ever be run as the root user. The normal convention is to set up *root* as an alias for the system administrator. This setting is a guard against slips in the configuration.

When a remote host connects to Exim in order to send mail, the only information Exim has about the host's identity is its IP address. The next configuration line,

```
host_lookup = *
```

specifies that Exim should do a reverse DNS lookup on all incoming connections, in order to get a host name. This improves the quality of the logging information, but if you feel it is too expensive, you can remove it entirely, or restrict the lookup to hosts on 'nearby' networks. Note that it is not always possible to find a host name from an IP address, because not all DNS reverse zones are maintained, and sometimes DNS servers are unreachable.

The next two lines are concerned with *ident* callbacks, as defined by RFC 1413 (hence their names):

```
rfc1413_hosts = *
rfc1413_query_timeout = 30s
```

These settings cause Exim to make ident callbacks for all incoming SMTP calls. You can limit the hosts to which these calls are made, or change the timeout that is used. If you set the timeout to zero, all ident calls are disabled. Although they are cheap and can provide useful information for tracing problem messages, some hosts and firewalls have problems with ident calls. This can result in a timeout instead of an immediate refused connection, leading to delays on starting up an incoming SMTP session.

When Exim receives messages over SMTP connections, it expects all addresses to be fully qualified with a domain, as required by the SMTP definition. However, if you are running a server to which simple clients submit messages, you may find that they send unqualified addresses. The two commented-out options:

```
# sender_unqualified_hosts =
# recipient_unqualified_hosts =
```

show how you can specify hosts that are permitted to send unqualified sender and recipient addresses, respectively.

The **percent_hack_domains** option is also commented out:

```
# percent_hack_domains =
```

It provides a list of domains for which the ‘percent hack’ is to operate. This is an almost obsolete form of explicit email routing. If you do not know anything about it, you can safely ignore this topic.

The last two settings in the main part of the default configuration are concerned with messages that have been ‘frozen’ on Exim’s queue. When a message is frozen, Exim no longer continues to try to deliver it. Freezing occurs when a bounce message encounters a permanent failure because the sender address of the original message that caused the bounce is invalid, so the bounce cannot be delivered. This is probably the most common case, but there are also other conditions that cause freezing, and frozen messages are not always bounce messages.

```
ignore_bounce_errors_after = 2d
timeout_frozen_after = 7d
```

The first of these options specifies that failing bounce messages are to be discarded after 2 days on the queue. The second specifies that any frozen message (whether a bounce message or not) is to be timed out (and discarded) after a week. In this configuration, the first setting ensures that no failing bounce message ever lasts a week.

7.2 ACL configuration

In the default configuration, the ACL section follows the main configuration. It starts with the line

```
begin acl
```

and it contains the definition of one ACL called *acl_check_rcpt* that was referenced in the setting of **acl_smtp_rcpt** above. This ACL is used for every RCPT command in an incoming SMTP message. Each RCPT command specifies one of the message’s recipients. The ACL statements are considered in order, until the recipient address is either accepted or rejected. The RCPT command is then accepted or rejected, according to the result of the ACL processing.

```
acl_check_rcpt:
```

This line, consisting of a name terminated by a colon, marks the start of the ACL, and names it.

```
accept  hosts = :
```

This ACL statement accepts the recipient if the sending host matches the list. But what does that strange list mean? It doesn’t actually contain any host names or IP addresses. The presence of the colon puts an empty item in the list; Exim matches this only if the incoming message didn’t come from a remote host. The colon is important. Without it, the list itself is empty, and can never match anything.

What this statement is doing is to accept unconditionally all recipients in messages that are submitted by SMTP from local processes using the standard input and output (that is, not using TCP/IP). A number of MUAs operate in this manner.

```
deny    local_parts   = ^.*[@%!/|] : ^\\.
```

This statement uses regular expressions to reject addresses with local parts that contain any of the characters ‘@’, ‘%’, ‘!’, ‘/’, and ‘|’, or that begin with dots. Although these characters are entirely legal in local parts (in the case of ‘@’ and leading dots, only if correctly quoted), they do not normally occur in Internet mail addresses.

The first three have in the past been associated with explicitly routed addresses (percent is still sometimes used – see the **percent_hack_domains** option). Addresses containing these characters are regularly tried by spammers in an attempt to bypass relaying restrictions, and also by open relay testing programs. Unless you really need them it is safest to reject these characters at this early stage. Slash and a leading dot are included in the list to avoid problems with local parts that are used to construct file names. Vertical bar is included because of its use as a pipe symbol in shell commands.

This configuration is heavy-handed in rejecting these characters for all messages it accepts from remote hosts. This is a deliberate policy of being as safe as possible. By moving the test elsewhere in

the ACL, you can relax the restriction so that, for example, it applies only to addresses in your local domains.

```
accept  local_parts   = postmaster
        domains       = +local_domains
```

This statement, which has two conditions, accepts an incoming address if the local part is *postmaster* and the domain is one of those listed in the *local_domains* domain list. The '+' character is used to indicate a reference to a named list. In this configuration, there is just one domain in *local_domains*, but in general there may be many.

The presence of this statement means that mail to postmaster is never blocked by any of the subsequent tests. This can be helpful while sorting out problems in cases where the subsequent tests are incorrectly denying access.

```
require verify        = sender
```

This statement requires the sender address to be verified before any subsequent ACL statement can be used. If verification fails, the incoming recipient address is refused. Verification consists of trying to route the address, to see if a bounce message could be delivered to it. In the case of remote addresses, basic verification checks only the domain, but *callouts* can be used for more verification if required. Section 37.13 discusses the details of address verification.

```
# deny    message       = rejected because $sender_host_address is \
#                         in a black list at $dnslist_domain\n\
#                         $dnslist_text
#         dnslists      = black.list.example
#
# warn    message       = X-Warning: $sender_host_address is \
#                         in a black list at $dnslist_domain
#         log_message   = found in $dnslist_domain
#         dnslists      = black.list.example
```

These commented-out lines are examples of how you could configure Exim to check sending hosts against a DNS black list. The first statement rejects messages from blacklisted hosts, whereas the second merely inserts a warning header line.

```
accept  domains       = +local_domains
        endpass
        message       = unknown user
        verify        = recipient
```

This statement accepts the incoming recipient address if its domain is one of the local domains, but only if the address can be verified. Verification of local addresses normally checks both the local part and the domain. The **endpass** line needs some explanation: if the condition above **endpass** fails, that is, if the address is not in a local domain, control is passed to the next ACL statement. However, if the condition below **endpass** fails, that is, if a recipient in a local domain cannot be verified, access is denied and the recipient is rejected. The **message** modifier provides a customized error message for the failure.

```
accept  domains       = +relay_to_domains
        endpass
        message       = unrouteable address
        verify        = recipient
```

This statement accepts the incoming recipient address if its domain is one of the domains for which this host is a relay, but again, only if the address can be verified.

```
accept  hosts         = +relay_from_hosts
```

Control reaches this statement only if the recipient's domain is neither a local domain, nor a relay domain. The statement accepts the address if the message is coming from one of the hosts that are defined as being allowed to relay through this host. Recipient verification is omitted here, because in

many cases the clients are dumb MUAs that do not cope well with SMTP error responses. If you are actually relaying out from MTAs, you should probably add recipient verification here.

```
accept  authenticated = *
```

Control reaches here for attempts to relay to arbitrary domains from arbitrary hosts. The statement accepts the address only if the client host has authenticated itself. The default configuration does not define any authenticators, which means that no client can in fact authenticate. You will need to add authenticator definitions if you want to make use of this ACL statement.

```
deny    message       = relay not permitted
```

The final statement denies access, giving a specific error message. Reaching the end of the ACL also causes access to be denied, but with the generic message ‘administrative prohibition’.

7.3 Router configuration

The router configuration comes next in the default configuration, introduced by the line

```
begin routers
```

Routers are the modules in Exim that make decisions about where to send messages. An address is passed to each router in turn, until it is either accepted, or failed. This means that the order in which you define the routers matters. Each router is fully described in its own chapter later in this manual. Here we give only brief overviews.

```
# domain_literal:
#   driver = ipliteral
#   domains = !+local_domains
#   transport = remote_smtp
```

This router is commented out because the vast majority of sites do not want to support domain literal addresses (those of the form *user@[10.9.8.7]*). If you uncomment this router, you will also need to uncomment the setting of **allow_domain_literals** in the main part of the configuration.

```
dnslookup:
  driver = dnslookup
  domains = ! +local_domains
  transport = remote_smtp
  ignore_target_hosts = 0.0.0.0 : 127.0.0.0/8
  no_more
```

The first uncommented router handles addresses that do not involve any local domains. This is specified by the line

```
domains = ! +local_domains
```

The **domains** option lists the domains to which this router applies, but the exclamation mark is a negation sign, so the router is used only for domains that are not in the domain list called *local_domains* (which was defined at the start of the configuration). The plus sign before *local_domains* indicates that it is referring to a named list. Addresses in other domains are passed on to the following routers.

The name of the router driver is **dnslookup**, and is specified by the **driver** option. Do not be confused by the fact that the name of this router instance is the same as the name of the driver. The instance name is arbitrary, but the name set in the **driver** option must be one of the driver modules that is in the Exim binary.

The **dnslookup** router routes addresses by looking up their domains in the DNS in order to obtain a list of hosts to which the address is routed. If the router succeeds, the address is queued for the **remote_smtp** transport, as specified by the **transport** option. If the router does not find the domain in the DNS, no further routers are tried because of the **no_more** setting, so the address fails and is bounced.

The **ignore_target_hosts** option specifies a list of IP addresses that are to be entirely ignored. This option is present because a number of cases have been encountered where MX records in the DNS point to host names whose IP addresses are 0.0.0.0 or are in the 127 subnet (typically 127.0.0.1). Completely ignoring these IP addresses causes Exim to fail to route the email address, so it bounces. Otherwise, Exim would log a routing problem, and continue to try to deliver the message periodically until the address timed out.

```
system_aliases:
  driver = redirect
  allow_fail
  allow_defer
  data = ${lookup{$local_part}lsearch{/etc/aliases}}
# user = exim
  file_transport = address_file
  pipe_transport = address_pipe
```

Control reaches this and subsequent routers only for addresses in the local domains. This router checks to see whether the local part is defined as an alias in the **/etc/aliases** file, and if so, redirects it according to the data that it looks up from that file. If no data is found for the local part, the value of the **data** option is empty, causing the address to be passed to the next router.

/etc/aliases is a conventional name for the system aliases file that is often used. That is why it is referenced by from the default configuration file. However, you can change this by setting SYSTEM_ALIASES_FILE in **Local/Makefile** before building Exim.

```
userforward:
  driver = redirect
  check_local_user
  file = $home/.forward
  no_verify
  no_expn
  check_ancestor
# allow_filter
  file_transport = address_file
  pipe_transport = address_pipe
  reply_transport = address_reply
```

This is the most complicated router in the default configuration. It is another redirection router, but this time it is looking for forwarding data set up by individual users. The **check_local_user** setting means that the first thing it does is to check that the local part of the address is the login name of a local user. If it is not, the router is skipped. When a local user is found, the file called **.forward** in the user's home directory is consulted. If it does not exist, or is empty, the router declines. Otherwise, the contents of **.forward** are interpreted as redirection data.

Traditional **.forward** files contain just a list of addresses, pipes, or files. Exim supports this by default. However, if **allow_filter** is set (it is commented out by default), the contents of the file are interpreted as a set of Exim filtering instructions, provided the file begins with '#Exim filter'. User filtering is discussed in the separate document entitled *Exim's interface to mail filtering*.

The **no_verify** and **no_expn** options mean that this router is skipped when verifying addresses, or when running as a consequence of an SMTP EXPN command. There are two reasons for doing this:

(1) Whether or not a local user has a **.forward** file is not really relevant when checking an address for validity; it makes sense not to waste resources doing unnecessary work.

(2) More importantly, when Exim is verifying addresses or handling an EXPN command during an SMTP session, it is running as the Exim user, not as root. It may therefore not be possible for Exim to read users' **.forward** files at this time.

The setting of **check_ancestor** prevents the router from generating a new address that is the same as any previous address that was redirected. (This works round a problem concerning a bad interaction between aliasing and forwarding – see section 21.5).

The final three option settings specify the transports that are to be used when forwarding generates a direct delivery to a file, or to a pipe, or sets up an auto-reply, respectively. For example, if a **.forward** file contains

```
a.nother@elsewhere.example, /home/spqr/archive
```

the delivery to **/home/spqr/archive** is done by running the **address_file** transport.

```
localuser:
  driver = accept
  check_local_user
  transport = local_delivery
```

The final router sets up delivery into local mailboxes, provided that the local part is the name of a local login, by accepting the address and queuing it for the **local_delivery** transport. Otherwise, we have reached the end of the routers, so the address is bounced.

7.4 Transport configuration

Transports define mechanisms for actually delivering messages. They operate only when referenced from routers, so the order in which they are defined does not matter. The transports section of the configuration starts with

```
begin transports
```

One remote transport and four local transports are defined.

```
remote_smtp:
  driver = smtp
```

This transport is used for delivering messages over SMTP connections. All its options are defaulted. The list of remote hosts comes from the router.

```
local_delivery:
  driver = appendfile
  file = /var/mail/$local_part
  delivery_date_add
  envelope_to_add
  return_path_add
# group = mail
# mode = 0660
```

This **appendfile** transport is used for local delivery to user mailboxes in traditional BSD mailbox format. By default it runs under the uid and gid of the local user, which requires the sticky bit to be set on the **/var/mail** directory. Some systems use the alternative approach of running mail deliveries under a particular group instead of using the sticky bit. The commented options show how this can be done.

Exim adds three headers to the message as it delivers it: *Delivery-date:*, *Envelope-to:* and *Return-path:*. This action is requested by the three similarly-named options above.

```
address_pipe:
  driver = pipe
  return_output
```

This transport is used for handling deliveries to pipes that are generated by redirection (aliasing or users' **.forward** files). The **return_output** option specifies that any output generated by the pipe is to be returned to the sender.

```
address_file:
  driver = appendfile
  delivery_date_add
  envelope_to_add
  return_path_add
```

This transport is used for handling deliveries to files that are generated by redirection. The name of the file is not specified in this instance of **appendfile**, because it comes from the **redirect** router.

```
address_reply:
  driver = autoreply
```

This transport is used for handling automatic replies generated by users' filter files.

7.5 Default retry rule

The retry section of the configuration file contains rules which affect the way Exim retries deliveries that cannot be completed at the first attempt. It is introduced by the line

```
begin retry
```

In the default configuration, there is just one rule, which applies to all errors:

```
*   *   F,2h,15m; G,16h,1h,1.5; F,4d,6h
```

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 6 hours up to 4 days. If an address is not delivered after 4 days of failure, it is bounced.

7.6 Rewriting configuration

The rewriting section of the configuration, introduced by

```
begin rewrite
```

contains rules for rewriting addresses in messages as they arrive. There are no rewriting rules in the default configuration file.

7.7 Authenticators configuration

The authenticators section of the configuration, introduced by

```
begin authenticators
```

defines mechanisms for the use of the SMTP AUTH command. No authenticators are specified in the default configuration file.

8. Regular expressions

Exim supports the use of regular expressions in many of its options. It uses the PCRE regular expression library; this provides regular expression matching that is compatible with Perl 5. The syntax and semantics of regular expressions is discussed in many Perl reference books, and also in Jeffrey Friedl's *Mastering Regular Expressions* (O'Reilly, ISBN 0-596-00289-0).

The documentation for the syntax and semantics of the regular expressions that are supported by PCRE is included in plain text in the file **doc/pcrepattern.txt** in the Exim distribution, and also in the HTML tarbundle of Exim documentation, and as an appendix to the Exim book. It describes in detail the features of the regular expressions that PCRE supports, so no further description is included here. The PCRE functions are called from Exim using the default option settings (that is, with no PCRE options set), except that the PCRE_CASELESS option is set when the matching is required to be case-insensitive.

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. After building Exim, the binary can be found in the build directory (it is not installed anywhere automatically). There is documentation of various options in **doc/pcretest.txt**, 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**.

9. File and database lookups

Exim can be configured to look up data in files or databases as it processes messages. Two different kinds of syntax are used:

(1) A string that is to be expanded may contain explicit lookup requests. These cause parts of the string to be replaced by data that is obtained from the lookup.

(2) Lists of domains, hosts, and email addresses can contain lookup requests as a way of avoiding excessively long linear lists. In this case, the data that is returned by the lookup is often (but not always) discarded; whether the lookup succeeds or fails is what really counts. These kinds of list are described in chapter 10.

It is easy to confuse the two different kinds of lookup, especially as the lists that may contain the second kind are always expanded before being processed as lists. Therefore, they may also contain lookups of the first kind. Be careful to distinguish between the following two examples:

```
domains = ${lookup{$sender_host_address}lsearch{/some/file}}
domains = lsearch;/some/file
```

The first uses a string expansion, the result of which must be a domain list. String expansions are described in detail in chapter 11. The expansion takes place first, and the file that is searched could contain lines like this:

```
192.168.3.4: domain1 : domain2 : ...
192.168.1.9: domain3 : domain4 : ...
```

Thus, the result of the expansion is a list of domains (and possibly other types of item that are allowed in domain lists).

In the second case, the lookup is a single item in a domain list. It causes Exim to use a lookup to see if the domain that is being processed can be found in the file. The file could contains lines like this:

```
domain1:
domain2:
```

Any data that follows the keys is not relevant when checking that the domain matches the list item.

It is possible to use both kinds of lookup at once. Consider a file containing lines like this:

```
192.168.5.6: lsearch;/another/file
```

If the value of **$sender_host_address** is 192.168.5.6, expansion of the first **domains** setting above generates the second setting, which therefore causes a second lookup to occur.

The rest of this chapter describes the different lookup types that are available. Any of them can be used in either of the circumstances described above. The syntax requirements for the two cases are described in chapters 11 and 10, respectively.

9.1 Lookup types

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 lookup type determines how the file is searched.
- The *query* style accepts a generalized database query. No particular key value is assumed by Exim for query-style lookups. You can use whichever Exim variable(s) you need to construct the database query.

The code for each lookup type is in a separate source file that is 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. For some types of lookup (e.g. SQL databases), you need to install appropriate libraries and header files before building Exim.

9.2 Single-key lookup types

The following single-key lookup types are implemented:

- **cdb**: The given file is searched as a Constant DataBase file, using the key string without a 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 in several places:

  http://www.pobox.com/~djb/cdb.html
  ftp://ftp.corpit.ru/pub/tinycdb/
  http://packages.debian.org/stable/utils/freecdb.html

  A cdb distribution is not needed in order to build Exim with cdb support, because 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, so you need to obtain a cdb distribution in order to do this.

- **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. A terminating binary zero is included in the key that is passed to the DBM library. See section 4.3 for a discussion of DBM libraries. For all versions of Berkeley DB, Exim uses the DB_HASH style of database when building DBM files using the **exim_dbmbuild** utility. However, when using Berkeley DB versions 3 or 4, it opens existing databases for reading with the DB_UNKNOWN option. This enables it to handle any of the types of database that the library supports, and can be useful for accessing DBM files created by other applications. (For earlier DB versions, DB_HASH is always used.)

- **dbmnz**: This is the same as **dbm**, except that a terminating binary zero is not included in the key that is passed to the DBM library. You may need this if you want to look up data in files that are created by or shared with some other application that does not use terminating zeros. For example, you need to use **dbmnz** rather than **dbm** if you want to authenticate incoming SMTP calls using the passwords from Courier's **/etc/userdbshadow.dat** file. Exim's utility program for creating DBM files (*exim_dbmbuild*) includes the zeros by default, but has an option to omit them (see section 45.8).

- **dsearch**: The given file must be a directory; this is searched for a file whose name is the key. The key may not contain any forward slash characters. The result of a successful lookup is the name of the file. An example of how this lookup can be used to support virtual domains is given in section 41.6.

- **lsearch**: The given file is a text file that is searched linearly for a line beginning with the key, terminated by a colon or white space or the end of the line. The first occurrence that is found in the file is used. 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, 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. Note that the keys in an **lsearch** file are literal strings. There is no wildcarding of any kind.

In most **lsearch** files, keys are not required to contain colons and whitespace. However, if you need this feature, it is available. If a key begins with a doublequote character, it is terminated only by a matching quote (or end of line), and the normal escaping rules apply to its contents (see section 6.12). An optional colon is permitted after quoted keys (exactly as for unquoted keys). There is no special handling of quotes for the data part of an **lsearch** line.

- **nis**: The given file is the name of a NIS map, and a NIS lookup is done with the given key, without a terminating binary zero. There is a variant called **nis0** which does include the terminating binary zero in the key. This is reportedly needed for Sun-style alias files. Exim does not recognize NIS aliases; the full map names must be used.

- **wildlsearch**: This searches a file linearly, like **lsearch**, but instead of being interpreted as a literal string, each key may be wildcarded. A general list-matching function is used, which means that each key is string-expanded as part of being tested. Like **lsearch**, the testing is done case-insensitively. The following forms of wildcard are recognized (in the expanded key):

  * The string may begin with an asterisk to mean 'begins with'. For example:

    ```
    *.a.b.c       data for anything.a.b.c
    *fish         data for anythingfish
    ```

  * The string may begin with a circumflex to indicate a regular expression. For example:

    ```
    ^\N\d+\.a\.b\N    data for <digits>.a.b
    ```

    Note the use of `\N` to disable expansion of the contents of the regular expression. If the regular expression contains white space or colon characters, you must either quote it (see **lsearch** above), or represent these characters in other ways. For example, `\s` can be used for white space and `\x3A` for a colon. This may be easier than quoting, because if you quote, you have to escape all the backslashes inside the quotes.

  * Although I cannot see it being of much use, the general matching function that is used to implement **wildlsearch** means that the string may begin with a lookup name terminated by a semicolon, and followed by lookup data. For example:

    ```
    cdb;/some/file  data for keys that match the file
    ```

    The data that is looked up is discarded.

  Keys that do not match any of these patterns are interpreted literally. The continuation rules for the data are the same as for **lsearch**, and keys may be followed by optional colons.

  Warning: Unlike other single-key lookup types, a file of data for **wildlsearch** can *not* be turned into a DBM or cdb file, because those lookup types support only literal keys.

9.3 Query-style lookup types

The supported query-style lookup types are listed below. Further details about many of them are given in later sections.

- **dnsdb**: This does a DNS search for a record whose domain name is the supplied query. The resulting data is the contents of the record. See section 9.9.

- **ldap**: This does an LDAP lookup using a query in the form of a URL, and returns attributes from a single entry. There is a variant called **ldapm** that permits values from multiple entries to be returned. A third variant called **ldapdn** returns the Distinguished Name of a single entry instead of any attribute values. See section 9.10.

- **mysql**: The format of the query is an SQL statement that is passed to a MySQL database. See section 9.17.

- **nisplus**: This does a NIS+ lookup using a query that can specify the name of the field to be returned. See section 9.16.

- **oracle**: The format of the query is an SQL statement that is passed to an Oracle database. See section 9.17.
- **passwd** is a query-style lookup with queries that are just user names. The lookup calls *getpwnam()* to interrogate the system password data, and on success, the result string is the same as you would get from an **lsearch** lookup on a traditional **/etc/passwd file**, though with * for the password value. For example:

  ```
  *:42:42:King Rat:/home/kr:/bin/bash
  ```
- **pgsql**: The format of the query is an SQL statement that is passed to a PostgreSQL database. See section 9.17.
- **testdb**: This is a lookup type that is used for testing Exim. It is not likely to be useful in normal operation.
- **whoson**: *Whoson* (**http://whoson.sourceforge.net**) is a proposed Internet protocol that allows Internet server programs to check whether a particular (dynamically allocated) IP address is currently allocated to a known (trusted) user and, optionally, to obtain the identity of the said user. In Exim, this can be used to implement 'POP before SMTP' checking using ACL statements such as

  ```
  require condition = \
    ${lookup whoson {$sender_host_address}{yes}{no}}
  ```

  The query consists of a single IP address. The value returned is the name of the authenticated user.

9.4 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. For this reason, it is not advisable to use a lookup that might do this for critical options such as a list of local domains.

When a lookup cannot be completed in a router or transport, delivery of the message (to the relevant address) is deferred, as for any other temporary error. In other circumstances Exim may assume the lookup has failed, or may give up altogether.

9.5 Default values in single-key lookups

In this context, a 'default value' is a value specified by the administrator that is to be used if a lookup fails.

If '∗' is added to a single-key lookup type (for example, **lsearch∗**) and 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 (or doesn't take place because there is no @ in the key), '∗' is looked up. For example, a **redirect** router might contain:

```
data = ${lsearch*@{$local_part@$domain}{/etc/mixed-aliases}}
```

Suppose the address that is being processed is *jane@eyre.example*. Exim looks up these keys, in this order:

```
jane@eyre.example
*@eyre.example
*
```

The data is taken from whichever key it finds first. **Note**: in an **lsearch** file, this does not mean the first of these keys in the file. A complete scan is done for each key, and only if it is not found at all does Exim move on to try the next key.

9.6 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 useful to be able to do partial matching. In this case, information in the file that has a key starting with '*.' is matched by any domain that ends with the components that follow the full stop. For example, if a key in a DBM file is

```
*.dates.fict.example
```

then when partial matching is enabled this is matched by (amongst others) *2001.dates.fict.example* and *1984.dates.fict.example*. It is also matched by *dates.fict.example*, if that does not appear as a separate key in the file.

Note: Partial matching is not available for query-style lookups. It is also not available for any lookup items in address lists (see section 10.13).

Partial matching is implemented by doing a series of separate lookups using keys constructed by modifying the original subject key. This means that it can be used with any of the single-key lookup types, provided that partial matching keys beginning with a special prefix (default '*.') are included in the data file. Keys in the file that do not begin with the prefix are matched only by unmodified subject keys when partial matching is in use.

Partial matching is requested by adding the string 'partial-' to the front of the name of a single-key lookup type, for example, **partial-dbm**. When this is done, the subject key is first looked up unmodified; if that fails, '*.' is added at the start of the subject key, and it is looked up again. If that fails, further lookups are tried with dot-separated components removed from the start of the subject key, one-by-one, and '*.' added on the front of what remains.

A minimum number of two non-* components are required. This 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 modified keys. Omitting the number is equivalent to 'partial2-'. If the subject key is *2250.dates.fict.example* then the following keys are looked up when the minimum number of non-* components is two:

```
2250.dates.fict.example
*.2250.dates.fict.example
*.dates.fict.example
*.fict.example
```

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

The use of '*.' as the partial matching prefix is a default that can be changed. The motivation for this feature is to allow Exim to operate with file formats that are used by other MTAs. A different prefix can be supplied in parentheses instead of the hyphen after 'partial'. For example:

```
domains = partial(.)lsearch;/some/file
```

In this example, if the domain is *a.b.c*, the sequence of lookups is `a.b.c`, `.a.b.c`, and `.b.c` (the default minimum of 2 non-wild components is unchanged). The prefix may consist of any punctuation characters other than a closing parenthesis. It may be empty, for example:

```
domains = partial1()cdb;/some/file
```

For this example, if the domain is *a.b.c*, the sequence of lookups is `a.b.c`, `b.c`, and `c`.

If 'partial0' is specified, what happens at the end (when the lookup with just one non-wild component has failed, and the original key is shortened right down to the null string) depends on the prefix:

- If the prefix has zero length, the whole lookup fails.
- If the prefix has length 1, a lookup for just the prefix is done. For example, the final lookup for 'partial0(.)' is for `.` alone.
- Otherwise, if the prefix ends in a dot, the dot is removed, and the remainder is looked up. With the default prefix, therefore, the final lookup is for '*' on its own.

- Otherwise, the whole prefix is looked up.

If the search type ends in ‘*’ or ‘*@’ (see section 9.5 above), the search for an ultimate default that this implies happens after all partial lookups have failed. If ‘partial0’ is specified, adding ‘*’ to the search type has no effect with the default prefix, because the ‘*’ key is already included in the sequence of partial lookups. However, there might be a use for lookup types such as ‘partial0(.)lsearch*’.

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.example` in a database file is useless, because the asterisk in a partial matching subject key is always followed by a dot.

9.7 Lookup caching

An Exim process 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 routing is complete.

9.8 Quoting lookup data

When data from an incoming message is included in a query-style lookup, there is the possibility of special characters in the data messing up the syntax of the query. For example, a NIS+ query that contains

```
[name=$local_part]
```

will be broken if the local part happens to contain a closing square bracket. For NIS+, data can be enclosed in double quotes like this:

```
[name="$local_part"]
```

but this still leaves the problem of a double quote in the data. The rule for NIS+ is that double quotes must be doubled. Other lookup types have different rules, and to cope with the differing requirements, an expansion operator of the following form is provided:

```
${quote_<lookup-type>:<string>}
```

For example, the safest way to write the NIS+ query is

```
[name="${quote_nisplus:$local_part}"]
```

See chapter 11 for full coverage of string expansions. The quote operator can be used for all lookup types, but has no effect for single-key lookups, since no quoting is ever needed in their key strings.

9.9 More about dnsdb

The **dnsdb** lookup type uses the DNS as its database. A query consists of a record type and a domain name, separated by an equals sign. For example, an expansion string could contain:

```
${lookup dnsdb{mx=a.b.example}{$value}fail}
```

The supported record types are A, CNAME, MX, NS, PTR, and TXT, and, when Exim is compiled with IPv6 support, AAAA (and A6 if that is also configured). If no type is given, TXT is assumed. When the type is PTR, the address should be given as normal; it is converted to the necessary inverted format internally. For example:

```
${lookup dnsdb{ptr=192.168.4.5}{$value}fail}
```

For MX records, both the preference value and the host name are returned, separated by a space. If multiple records are found (or, for A6 lookups, if a single record leads to multiple addresses), the data is returned as a concatenation, separated by newlines. The order, of course, depends on the DNS resolver.

9.10 More about LDAP

The original LDAP implementation came from the University of Michigan; this has become 'Open LDAP', and there are now two different releases. Another implementation comes from Netscape, and Solaris 7 and subsequent releases contain 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=OPENLDAP1
LDAP_LIB_TYPE=OPENLDAP2
LDAP_LIB_TYPE=NETSCAPE
LDAP_LIB_TYPE=SOLARIS
```

If LDAP_LIB_TYPE is not set, Exim assumes `OPENLDAP1`, which has the same interface as the University of Michigan version.

There are three LDAP lookup types in Exim. These behave slightly differently in the way they handle the results of a query:

- **ldap** requires the result to contain just one entry; if there are more, it gives an error.
- **ldapdn** also requires the result to contain just one entry, but it is the Distinguished Name that is returned rather than any attribute values.
- **ldapm** permits the result to contain more than one entry; the attributes from all of them are returned.

For **ldap** and **ldapm**, if a query finds only entries with no attributes, Exim behaves as if the entry did not exist, and the lookup fails. The format of the data returned by a successful lookup is described in the next section. First we explain how LDAP queries are coded.

9.11 Format of LDAP queries

An LDAP query takes the form of a URL as defined in RFC 2255. For example, in the configuration of a **redirect** router one might have this setting:

```
data = ${lookup ldap \
  {ldap:///cn=$local_part,o=University%20of%20Cambridge,\
  c=UK?mailbox?base?}}
```

The URL may begin with `ldap` or `ldaps` if your LDAP library supports secure (encrypted) LDAP connections. The second of these ensures that an encrypted TLS connection is used.

9.12 LDAP quoting

Two levels of quoting are required in LDAP queries, the first for LDAP itself and the second because the LDAP query is represented as a URL. Furthermore, within an LDAP query, two different kinds of quoting are required. For this reason, there are two different LDAP-specific quoting operators.

The **quote_ldap** operator is designed for use on strings that are part of filter specifications. Conceptually, it first does the following conversions on the string:

```
*    =>    \2A
(    =>    \28
)    =>    \29
\    =>    \5C
```

in accordance with RFC 2254. The resulting string is then quoted according to the rules for URLs, that is, all characters except

```
! $ ' - . _ ( ) * +
```

are converted to their hex values, preceded by a percent sign. For example:

```
${quote_ldap: a(bc)*, a<yz>; }
```

yields

```
%20a%5C28bc%5C29%5C2A%2C%20a%3Cyz%3E%3B%20
```

Removing the URL quoting, this is (with a leading and a trailing space):

```
a\28bc\29\2A, a<yz>;
```

The **quote_ldap_dn** operator is designed for use on strings that are part of base DN specifications in queries. Conceptually, it first converts the string by inserting a backslash in front of any of the following characters:

```
, + " \ < > ;
```

It also inserts a backslash before any leading spaces or # characters, and before any trailing spaces. (These rules are in RFC 2253.) The resulting string is then quoted according to the rules for URLs. For example:

```
${quote_ldap_dn: a(bc)*, a<yz>; }
```

yields

```
%5C%20a(bc)*%5C%2C%20a%5C%3Cyz%5C%3E%5C%3B%5C%20
```

Removing the URL quoting, this is (with a trailing space):

```
\ a(bc)*\, a\<yz\>\;\
```

There are some further comments about quoting in the section on LDAP authentication below.

9.13 LDAP connections

The connection to an LDAP server may either be over TCP/IP, or, when OpenLDAP is in use, via a Unix domain socket. The example given above does not specify an LDAP server. A server that is reached by TCP/IP can be specified in a query by starting it with

```
ldap://<hostname>:<port>/...
```

If the port (and preceding colon) are omitted, the standard LDAP port (389) is used. When no server is specified in a query, a list of default servers is taken from the **ldap_default_servers** configuration option. This supplies a colon-separated list of servers which are tried in turn until one successfully handles a query, or there is a serious error. Successful handling either returns the requested data, or indicates that it does not exist. Serious errors are syntactical, or multiple values when only a single value is expected. Errors which cause the next server to be tried are connection failures, bind failures, and timeouts.

For each server name in the list, a port number can be given. The standard way of specifing a host and port is to use a colon separator (RFC 1738). Because **ldap_default_servers** is a colon-separated list, such colons have to be doubled. For example

```
ldap_default_servers = ldap1.example.com::145:ldap2.example.com
```

If **ldap_default_servers** is unset, a URL with no server name is passed to the LDAP library with no server name, and the library's default (normally the local host) is used.

If you are using the OpenLDAP library, you can connect to an LDAP server using a Unix domain socket instead of a TCP/IP connection. This is specified by using `ldapi` instead of `ldap` in LDAP queries. What follows here applies only to OpenLDAP. If Exim is compiled with a different LDAP library, this feature is not available.

For this type of connection, instead of a host name for the server, a pathname for the socket is required, and the port number is not relevant. The pathname can be specified either as an item in **ldap_default_servers**, or inline in the query. In the former case, you can have settings such as

```
ldap_default_servers = /tmp/ldap.sock : backup.ldap.your.domain
```

When the pathname is given in the query, you have to escape the slashes as `%2F` to fit in with the LDAP URL syntax. For example:

```
${lookup ldap {ldapi://%2Ftmp%2Fldap.sock/o=...
```

When Exim processes an LDAP lookup and finds that the 'hostname' is really a pathname, it uses the Unix domain socket code, even if the query actually specifies `ldap` or `ldaps`. In particular, no encryption is used for a socket connection. This behaviour means that you can use a setting of **ldap_default_servers** such as in the example above with traditional `ldap` or `ldaps` queries, and it will work. First, Exim tries a connection via the Unix domain socket; if that fails, it tries a TCP/IP connection to the backup host.

If an explicit `ldapi` type is given in a query when a host name is specified, an error is diagnosed. However, if there are more items in **ldap_default_servers**, they are tried. In other words:

- Using a pathname with `ldap` or `ldaps` forces the use of the Unix domain interface.
- Using `ldapi` with a host name causes an error.

Using `ldapi` with no host or path in the query, and no setting of **ldap_default_servers**, does whatever the library does by default.

9.14 LDAP authentication and control information

The LDAP URL syntax provides no way of passing authentication and other control information to the server. To make this possible, the URL in an LDAP query may be preceded by any number of '*<name>*=*<value>*' settings, separated by spaces. If a value contains spaces it must be enclosed in double quotes, and when double quotes are used, backslash is interpreted in the usual way inside them.

The following names are recognized:

`CONNECT`	set a connection timeout
`USER`	set the DN, for authenticating the LDAP bind
`PASS`	set the password, likewise
`SIZE`	set the limit for the number of entries returned
`TIME`	set the maximum waiting time for a query

Here is an example of an LDAP query in an Exim lookup that uses some of these values. This is a single line, folded for ease of reading:

```
${lookup ldap
  {user="cn=manager,o=University of Cambridge,c=UK" pass=secret
  ldap:///o=University%20of%20Cambridge,c=UK?sn?sub?(cn=foo)}
  {$value}fail}
```

The encoding of spaces as %20 is a URL thing which should not be done for any of the auxiliary data. Exim configuration settings that include lookups which contain password information should be preceded by 'hide' to prevent non-admin users from using the **-bP** option to see their values.

The auxiliary data items may be given in any order. The default is no connection timeout (the system timeout is used), no user or password, no limit on the number of entries returned, and no time limit on queries.

The time limit for connection is given in seconds; zero means use the default. This facility is available in Netscape SDK 4.1; it may not be available in other LDAP implementations. Exim uses the given value if LDAP_X_OPT_CONNECT_TIMEOUT is defined in the LDAP headers.

When a DN is quoted in the USER= setting for LDAP authentication, Exim removes any URL quoting that it may contain before passing it LDAP. Apparently some libraries do this for themselves, but some do not. Removing the URL quoting has two advantages:

- It makes it possible to use the same **quote_ldap_dn** expansion for USER= DNs as with DNs inside actual queries.
- It permits spaces inside USER= DNs.

For example, a setting such as

```
USER=cn=${quote_ldap_dn:$1}
```

should work even if **$1** contains spaces.

Expanded data for the PASS= value should be quoted using the **quote** expansion operator, rather than the LDAP quote operators. The only reason this field needs quoting is to ensure that it conforms to the Exim syntax, which does not allow unquoted spaces. For example:

```
PASS=${quote:$3}
```

The LDAP authentication mechanism can be used to check passwords as part of SMTP authentication. See the **ldapauth** expansion string condition in chapter 11.

9.15 Format of data returned by LDAP

The **ldapdn** lookup type returns the Distinguished Name from a single entry as a sequence of values, for example

```
cn=manager, o=University of Cambridge, c=UK
```

The **ldap** lookup type generates an error if more than one entry matches the search filter, whereas **ldapm** permits this case, and inserts a newline in the result between the data from different entries. It is possible for multiple values 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.

In the common case where you specify a single attribute in your LDAP query, the result is not quoted, and does not contain the attribute name. If the attribute has multiple values, they are separated by commas.

If you specify multiple attributes, the result contains space-separated, quoted strings, each preceded by the attribute name and an equals sign. Within the quotes, the quote character, backslash, and newline are escaped with backslashes, and commas are used to separate multiple values for the attribute. Apart from the escaping, the string within quotes takes the same form as the output when a single attribute is requested. Specifying no attributes is the same as specifying all of an entry's attributes.

Here are some examples of the output format. The first line of each pair is an LDAP query, and the second is the data that is returned. The attribute called **attr1** has two values, whereas **attr2** has only one value:

```
ldap:///o=base?attr1?sub?(uid=fred)
value1.1, value1.2

ldap:///o=base?attr2?sub?(uid=fred)
value two

ldap:///o=base?attr1,attr2?sub?(uid=fred)
attr1="value1.1, value1.2" attr2="value two"

ldap:///o=base??sub?(uid=fred)
objectClass="top" attr1="value1.1, value1.2" attr2="value two"
```

The **extract** operator in string expansions can be used to pick out individual fields from data that consists of *key=value* pairs. You can make use of Exim's **-be** option to run expansion tests and thereby check the results of LDAP lookups.

9.16 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. The effect of the **quote_nisplus** expansion operator is to double any quote characters within the text.

9.17 More about MySQL, PostgreSQL, and Oracle

If any MySQL, PostgreSQL, or Oracle lookups are used, the **mysql_servers**, **pgsql_servers**, or **oracle_servers** option (as appropriate) must be set to a colon-separated list of server information. Each item in the list is a slash-separated list of four items: host name, database name, user name, and password. In the case of Oracle, the host name field is used for the 'service name', and the database name field is not used and should be empty. For example:

```
hide oracle_servers = oracle.plc.example//ph10/abcdwxyz
```

Because password data is sensitive, you should always precede the setting with 'hide', to prevent non-admin users from obtaining the setting via the **-bP** option. Here is an example where two MySQL servers are listed:

```
hide mysql_servers = localhost/users/root/secret:\
                     otherhost/users/root/othersecret
```

For MySQL and PostgreSQL, a host may be specified as <*name*>:<*port*> but because this is a colon-separated list, the colon has to be doubled.

For each query, these parameter groups are tried in order until a connection and a query succeeds. Queries for these databases are SQL statements, so an example might be

```
${lookup mysql{select mailbox from users where id='ph10'}{$value}fail}
```

If the result of the query contains more than one field, the data for each field in the row is returned, preceded by its name, so the result of

```
${lookup pgsql{select home,name from users where id='ph10'}{$value}}
```

might be

```
home=/home/ph10 name="Philip Hazel"
```

Values containing spaces and empty values are double quoted, with embedded quotes escaped by a backslash.

If the result of the query contains just one field, the value is passed back verbatim, without a field name, for example:

```
Philip Hazel
```

If the result of the query yields more than one row, it is all concatenated, with a newline between the data for each row.

The **quote_mysql**, **quote_pgsql**, and **quote_oracle** expansion operators convert newline, tab, carriage return, and backspace to \n, \t, \r, and \b respectively, and the characters single-quote, double-quote, and backslash itself are escaped with backslashes. The **quote_pgsql** expansion operator, in addition, escapes the percent and underscore characters. This cannot be done for MySQL because these escapes are not recognized in contexts where these characters are not special.

9.18 Special MySQL features

For MySQL, an empty host name or the use of 'localhost' in **mysql_servers** causes a connection to the server on the local host by means of a Unix domain socket. An alternate socket can be specified in parentheses. The full syntax of each item in **mysql_servers** is:

<hostname>`::`*<port>*`(`*<socket name>*`)/`*<database>*`/`*<user>*`/`*<password>*

Any of the three sub-parts of the first field can be omitted. For normal use on the local host it can be left blank or set to just 'localhost'.

No database need be supplied – but if it is absent here, it must be given in the queries.

If a MySQL query is issued that does not request any data (an insert, update, or delete command), the result of the lookup is the number of rows affected.

9.19 Special PostgreSQL features

PostgreSQL lookups can also use Unix domain socket connections to the database. This is usually faster and costs less CPU time than a TCP/IP connection. However it can be used only if the mail server runs on the same machine as the database server. A configuration line for PostgreSQL via Unix domain sockets looks like this:

```
hide pgsql_servers = (/tmp/.s.PGSQL.5432)/db/user/password : ...
```

In other words, instead of supplying a host name, a path to the socket is given. The path name is enclosed in parentheses so that its slashes aren't visually confused with the delimiters for the other server parameters.

If a PostgreSQL query is issued that does not request any data (an insert, update, or delete command), the result of the lookup is the number of rows affected.

10. Domain, Host, Address, and Local Part lists

A number of Exim configuration options contain lists of domains, hosts, email addresses, or local parts. For example, the **hold_domains** option contains a list of domains whose delivery is currently suspended. These lists are also used as data in ACL statements (see chapter 37).

Each item in one of these lists is a pattern to be matched against a domain, host, email address, or local part, respectively. In the sections below, the different types of pattern for each case are described, but first we cover some general facilities that apply to all four kinds of list.

10.1 Expansion of lists

Each list is expanded as a single string before it is used. If the expansion is forced to fail, Exim behaves as if the item it is testing (domain, host, address, or local part) is not in the list. Other expansion failures cause temporary errors.

If an item in a list is a regular expression, backslashes, dollars and possibly other special characters in the expression must be protected against misinterpretation by the string expander. The easiest way to do this is to use the `\N` expansion feature to indicate that the contents of the regular expression should not be expanded. For example, in an ACL you might have:

```
deny senders = \N^\d{8}\w@.*\.baddomain\.example$\N :
               ${lookup{$domain}lsearch{/badsenders/bydomain}}
```

The first item is a regular expression that is protected from expansion by `\N`, whereas the second uses the expansion to obtain a list of unwanted senders based on the receiving domain.

After expansion, the list is split up into separate items for matching. Normally, colon is used as the separator character, but this can be varied if necessary, as described in section 6.15.

10.2 Negated items in lists

Items in a list may be positive or negative. Negative items are indicated by a leading exclamation mark, which may be followed by optional white space. A list defines a set of items (domains, etc). When Exim processes one of these lists, it is trying to find out whether a domain, host, address, or local part (respectively) is in the set that is defined by the list. It works like this:

The list is scanned from left to right. If a positive item is matched, the subject that is being checked is in the set; if a negative item is matched, the subject is not in the set. If the end of the list is reached without the subject having matched any of the patterns, it is in the set if the last item was a negative one, but not if it was a positive one. For example, the list in

```
domainlist 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* do not match, because the last item in the list is positive. However, if the setting were

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

then all domains other than *a.b.c* would match because the last item in the list is negative. In other words, a list that ends with a negative item behaves as if it had an extra item `:*` on the end.

Another way of thinking about positive and negative items in lists is to read the connector as ‘or’ after a positive item and as ‘and’ after a negative item.

10.3 File names in lists

If an item in a domain, host, address, or local part list is an absolute file name (beginning with a slash character), each line of the file is read and processed as if it were an independent item in the list, except that further file names are not allowed, and no expansion of the data from the file takes place. Empty lines in the file are ignored, and the file may also contain comment lines:

- For domain and host lists, if a # character appears anywhere in a line of the file, it and all following characters are ignored.
- Because local parts may legitimately contain # characters, a comment in an address list or local part list file is recognized only if # is preceded by white space or the start of the line. For example:

```
not#comment@x.y.z   # but this is a comment
```

Putting a file name in a list has the same effect as inserting each line of the file as an item in the list (blank lines and comments excepted). However, there is one important difference: the file is read each time the list is processed, so if its contents vary over time, Exim's behaviour changes.

If a 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.

10.4 An lsearch file is not an out-of-line list

As will be described in the sections that follow, lookups can be used in lists to provide indexed methods of checking list membership. There has been some confusion about the way **lsearch** lookups work in lists. Because an **lsearch** file contains plain text and is scanned sequentially, it is sometimes thought that it is allowed to contain wild cards and other kinds of non-constant pattern. This is not the case. The keys in an **lsearch** file are always fixed strings, just as for any other single-key lookup type.

If you want to use a file to contain wild-card patterns that form part of a list, just give the file name on its own, without a search type, as described in the previous section.

10.5 Named lists

A list of domains, hosts, email addresses, or local parts can be given a name which is then used to refer to the list elsewhere in the configuration. This is particularly convenient if the same list is required in several different places. It also allows lists to be given meaningful names, which can improve the readability of the configuration. For example, it is conventional to define a domain list called *local_domains* for all the domains that are handled locally on a host, using a configuration line such as

```
domainlist local_domains = localhost:my.dom.example
```

Named lists are referenced by giving their name preceded by a plus sign, so, for example, a router that is intended to handle local domains would be configured with the line

```
domains = +local_domains
```

The first router in a configuration is often one that handles all domains except the local ones, using a configuration with a negated item like this:

```
dnslookup:
  driver = dnslookup
  domains = ! +local_domains
  transport = remote_smtp
  no_more
```

The four kinds of named list are created by configuration lines starting with the words **domainlist**, **hostlist**, **addresslist**, or **localpartlist**, respectively. Then there follows the name that you are defining, followed by an equals sign and the list itself. For example:

```
hostlist    relay_hosts = 192.168.23.0/24 : my.friend.example
addresslist bad_senders = cdb;/etc/badsenders
```

A named list may refer to other named lists:

```
domainlist  dom1 = first.example : second.example
domainlist  dom2 = +dom1 : third.example
domainlist  dom3 = fourth.example : +dom2 : fifth.example
```

Warning: If the last item in a referenced list is a negative one, the effect may not be what you intended, because the negation does not propagate out to the higher level. For example, consider:

```
domainlist  dom1 = !a.b
domainlist  dom2 = +dom1 : *.b
```

The second list specifies ‘either in the **dom1** list or **.b*’. The first list specifies just ‘not *a.b*’, so the domain *x.y* matches it. That means it matches the second list as well. The effect is not the same as

```
domainlist  dom2 = !a.b : *.b
```

where *x.y* does not match. It’s best to avoid negation altogether in referenced lists if you can.

Named lists may have a performance advantage. When Exim is routing an address or checking an incoming message, it caches the result of tests on named lists. So, if you have a setting such as

```
domains = +local_domains
```

on several of your routers or in several ACL statements, the actual test is done only for the first one. However, the caching works only if there are no expansions within the list itself or any sublists that it references. In other words, caching happens only for lists that are known to be the same each time they are referenced.

By default, there may be up to 16 named lists of each type. This limit can be extended by changing a compile-time variable. The use of domain and host lists is recommended for concepts such as local domains, relay domains, and relay hosts. The default configuration is set up like this.

10.6 Named lists compared with macros

At first sight, named lists might seem to be no different from macros in the configuration file. However, macros are just textual substitutions. If you write

```
ALIST = host1 : host2
auth_advertise_hosts = !ALIST
```

it probably won’t do what you want, because that is exactly the same as

```
auth_advertise_hosts = !host1 : host2
```

Notice that the second host name is not negated. However, if you use a host list, and write

```
hostlist alist = host1 : host2
auth_advertise_hosts = ! +alist
```

the negation applies to the whole list, and so that is equivalent to

```
auth_advertise_hosts = !host1 : !host2
```

10.7 Domain lists

Domain lists contain patterns that are to be matched against a mail domain. The following types of item may appear in domain lists:

- If a pattern consists of a single @ character, it matches the local host name, as set by the **primary_hostname** option (or defaulted). This makes it possible to use the same configuration file on several different hosts that differ only in their names.
- If a pattern consists of the string `@[]` it matches any local IP interface address, enclosed in square brackets, as in an email address that contains a domain literal. The use of domain literals is dying out in today's Internet.
- If a pattern consists of the string `@mx_any` it matches any domain that has an MX record pointing to the local host or to any host that is listed in **hosts_treat_as_local**. The items `@mx_primary` and `@mx_secondary` are similar, except that the first matches only when a primary MX target is the local host, and the second only when no primary MX target is the local host, but a secondary MX target is. 'Primary' means an MX record with the lowest preference value – there may of course be more than one of them.
- If a pattern starts with an asterisk, 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, 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.

  Warning: Because domain lists are expanded before being processed, you must escape any backslash and dollar characters in the regular expression, or use the special `\N` sequence (see chapter 11) to specify that it is not to be expanded (unless you really do want to build a regular expression by expansion, of course).
- If a pattern starts with the name of a single-key lookup type followed by a semicolon (for example, 'dbm;' or 'lsearch;'), the remainder of the pattern must be a file name in a suitable format for the lookup type. For example, for 'cdb;' it must be an absolute path:

  ```
  domains = cdb;/etc/mail/local_domains.cdb
  ```

  The appropriate type of lookup is done on the file using the domain name as the key. In most cases, the data that is looked up is not used; Exim is interested only in whether or not the key is present in the file. However, when a lookup is used for the **domains** option on a router or a **domains** condition in an ACL statement, the data is preserved in the **$domain_data** variable and can be referred to in other router options or other statements in the same ACL.
- Any of the single-key lookup type names may be preceded by 'partial<*n*>-', where the <*n*> is optional, for example,

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

  This causes partial matching logic to be invoked; a description of how this works is given in section 9.6.
- 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;'), the remainder of the pattern must be an appropriate query for the lookup type, as described in chapter 9. For example:

  ```
  hold_domains = mysql;select domain from holdlist \
    where domain = '$domain';
  ```

In most cases, the data that is looked up is not used (so for an SQL query, for example, it doesn't matter what field you select). Exim is interested only in whether or not the query succeeds. However, when a lookup is used for the **domains** option on a router, the data is preserved in the **$domain_data** variable and can be referred to in other options.

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

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

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

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.

10.8 Host lists

Host lists are used to control what remote hosts are allowed to do. For example, some hosts may be allowed to use the local host as a relay, and some may be permitted to use the SMTP ETRN command. Hosts can be identified in two different ways, by name or by IP address. In a host list, some types of pattern are matched to a host name, and some are matched to an IP address. You need to be particularly careful with this when single-key lookups are involved, to ensure that the right value is being used as the key.

10.9 Special host list patterns

If a host list item is the empty string, it matches only when no remote host is involved. This is the case when a message is being received from a local process using SMTP on the standard input, that is, when a TCP/IP connection is not used.

The special pattern '*' in a host list matches any host or no host. Neither the IP address nor the name is actually inspected.

10.10 Host list patterns that match by IP address

If an IPv4 host calls an IPv6 host and the call is accepted on an IPv6 socket, 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. (Not all operating systems accept IPv4 calls on IPv6 sockets, as there have been some security concerns.)

The following types of pattern in a host list check the remote host by inspecting its IP address:

- If the pattern is a plain domain name (not a regular expression, not starting with *), Exim calls the operating system function to find the associated IP address(es). Exim uses the newer *getipnodebyname()* function when available, otherwise *gethostbyname()*. This typically causes a forward DNS lookup of the name. The result is compared with the IP address of the subject host.
- If the pattern is '@', the primary host name is substituted and used as a domain name, as just described.
- If the pattern is an IP address, it is matched against the IP address of the subject host. IPv4 addresses are given in the normal 'dotted-quad' notation. IPv6 addresses can be given in colon-separated format, but the colons have to be doubled so as not to be taken as item separators when

the default list separator is used. IPv6 addresses are recognized even when Exim is compiled without IPv6 support. This means that if they appear in a host list on an IPv4-only host, Exim will not treat them as host names. They are just addresses that can never match a client host.

- If the pattern is '@[]', it matches the IP address of any IP interface on the local host. For example, if the local host is an IPv4 host with one interface address 10.45.23.56, these two ACL statements have the same effect:

```
accept hosts = 127.0.0.1 : 10.45.23.56
accept hosts = @[]
```

- If the pattern is an IP address followed by a slash and a mask length (for example 10.11.42.0/24), it is matched against the IP address of the subject host under the given mask. This allows, an entire network of hosts to be included (or excluded) by a single item. The mask uses CIDR notation; it specifies the number of address bits that must match, starting from the most significant end of the address.

  Note: the mask is *not* a count of addresses, nor is it the high number of a range of addresses. It is the number of bits in the network portion of the address. The above example specifies a 24-bit netmask, so it matches all 256 addresses in the 10.11.42.0 network. An item such as

```
192.168.23.236/31
```

  matches just two addresses, 192.168.23.236 and 192.168.23.237. A mask value of 32 for an IPv4 address is the same as no mask at all; just a single address matches.

  Here is another example which shows an IPv4 and an IPv6 network:

```
recipient_unqualified_hosts = 192.168.0.0/16: \
                              3ffe::ffff::836f::::/48
```

  The doubling of list separator characters applies only when these items appear inline in a host list. It is not required when indirecting via a file. For example,

```
recipient_unqualified_hosts = /opt/exim/unqualnets
```

  could make use of a file containing

```
172.16.0.0/12
3ffe:ffff:836f::/48
```

  to have exactly the same effect as the previous example. When listing IPv6 addresses inline, it is usually more convenient to use the facility for changing separator characters. This list contains the same two networks:

```
recipient_unqualified_hosts = <; 172.16.0.0/12; \
                                 3ffe:ffff:836f::/48
```

  The separator is changed to semicolon by the leading '<;' at the start of the list.

- When a host is to be identified by a lookup of its complete IP address, the pattern can take this form:

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

  For example:

```
hosts_lookup = net-cdb;/hosts-by-ip.db
```

  For a single-key lookup type, the text form of the IP address of the subject host is used as the lookup key. IPv6 addresses are converted to an unabbreviated form, using lower case letters, with dots as separators because colon is the key terminator in **lsearch** files. [Colons can in fact be used in keys in **lsearch** files by quoting the keys, but this is a facility that was added later.]

  For a query-style lookup, the variable **$sender_host_address** can be used in the query. For example:

```
hosts_lookup = net-pgsql;\
  select ip from hostlist where ip='$sender_host_address'
```

The value of **$sender_host_address** for an IPv6 address uses colon separators. You can use the **sg** expansion item to change this if you need to.

In both cases, the data returned by the lookup is not used.

- Lookups can also be performed using masked IP addresses, using patterns of this form:

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

  For example:

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

  If the search type is a single-key lookup, the IP address of the subject host is masked using <*number*> as the mask length. A textual string is constructed from the masked value, followed by the mask, and this is used as the lookup key. For example, if the host's IP address is 192.168.34.6, the key that is looked up for the above example is '192.168.34.0/24'. IPv6 addresses are converted to a text value using lower case letters and dots as separators instead of the more usual colon, because colon is the key terminator in **lsearch** files. Full, unabbreviated IPv6 addresses are always used.

  Warning: Specifing **net32-** (for an IPv4 address) or **net128-** (for an IPv6 address) is not the same as specifing just **net-** without a number. In the former case the key strings include the mask value, whereas in the latter case the IP address is used on its own.

  If the search type is a query-style lookup, <*number*> is not relevant, and this type of pattern is no different to the previous kind, that is, **net-** without a number. If you want to use masked IP addresses in database queries, you can use the **mask** expansion operator.

10.11 Host list patterns that match by host name

The remaining types of pattern that can appear in host lists require Exim to know the name of the remote host. They are all wildcarded names of different kinds. (If a complete name is given without any wildcarding, it is used to find an IP address to match against, as described in the previous section.)

If the remote host name is not already known when Exim encounters one of these patterns, a system function (*gethostbyaddr()* or *getipnodebyaddr()* if available) is used to find it from the IP address. This typically causes a reverse DNS lookup to occur. Although many sites on the Internet are conscientious about maintaining reverse DNS data for their hosts, there are also many that do not do this. Consequently, a name cannot always be found, and this may lead to unwanted effects.

If the DNS lookup fails, that is, if there is no reverse DNS entry for the IP address, Exim behaves as if the host does not match the list. This may not always be what you want to happen. To change Exim's behaviour, the special item '+include_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, but no name can be found, Exim behaves as if the host does match the list. For example,

```
host_reject_connection = +include_unknown:*.enemy.ex
```

rejects connections from any host whose name matches `*.enemy.ex`, and also any hosts whose name it cannot find.

Warning: Take care when configuring host lists with wildcarded name patterns. Consider what will happen if a name cannot be found.

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

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

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

  is a regular expression which matches either of the two hosts *a.c.d* or *b.c.d*. When a regular expression is used in a host list, you must take care that backslash and dollar characters are not misinterpreted as part of the string expansion. The simplest way to do this is to use `\N` to mark that part of the string as non-expandable. For example:

  ```
  sender_unqualified_hosts = \N^(a|b)\.c\.d$\N : ....
  ```

- If a pattern is of the form

  <search-type>;<filename or query>

  for example

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

  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.

  Warning 1: When using this kind of pattern with a single-key lookup, you must have host *names* as keys in the file, not IP addresses. If you want to do lookups based on IP addresses, you must precede the search type with 'net-' (see the previous section). There is, however, no reason why you could not use two items in the same list, one doing an address lookup and one doing a name lookup, both using the same file.

  Warning 2: For a query-style lookup, what to lookup is given explicitly in the query, but Exim always ensures that the host name is available before running the query for this type of pattern. If you are not using the host name in your query, you should be using the `net-` form of search described in the previous section, so that Exim does not look up the host name unnecessarily.

10.12 Mixing wildcarded host names and addresses in host lists

If you have name lookups or wildcarded host names and IP addresses in the same host list, you should normally put the IP addresses first. For example, in an ACL you could have:

```
accept hosts = 10.9.8.7 : *.friend.example
```

The reason for this lies in the left-to-right way that Exim processes lists. It can test IP addresses without doing any DNS lookups, but when it reaches an item that requires a host name, it fails if it cannot find a host name to compare with the pattern. If the above list is given in the opposite order, the **accept** statement fails for a host whose name cannot be found, even if its IP address is 10.9.8.7.

If you really do want to do the name check first, and still recognize the IP address, you can rewrite the ACL like this:

```
accept hosts = *.friend.example
accept hosts = 10.9.8.7
```

If the first **accept** fails, Exim goes on to try the second one. See chapter 37 for details of ACLs.

10.13 Address lists

Address lists contain patterns that are matched against mail addresses. There is one special case to be considered: the sender address of a bounce message is always empty. You can test for this by providing an empty item in an address list. For example, you can set up a router to process bounce messages by using this option setting:

```
senders = :
```

The presence of the colon creates an empty item. If you do not provide any data, the list is empty and matches nothing. The empty sender can also be detected by a regular expression that matches an empty string.

The following kinds of pattern are supported in address lists:

- If (after expansion) a pattern starts with '^', a regular expression match is done against the complete address, with the pattern as the regular expression. You must take care that backslash and dollar characters are not misinterpreted as part of the string expansion. The simplest way to do this is to use `\N` to mark that part of the string as non-expandable. For example:

  ```
  deny senders = \N^\d{8}.+@spamhaus.example$\N : ...
  ```

  The `\N` sequences are removed by the expansion, so the item does start with '^' by the time it is being interpreted as an address pattern.

- If a pattern starts with '@@' followed by a single-key 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 '*' (see section 9.5). 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

  ```
  deny senders = @@dbm;/etc/reject-by-domain
  ```

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

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

  to reject all senders except **postmaster** from that domain. 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, 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.

- The @@<*lookup*> style of item can also be used with a query-style lookup, but in this case, the chaining facility is not available. The lookup can only return a single list of local parts.

- Complete addresses can be looked up by using a pattern that starts with a lookup type terminated by a semicolon, follwed by the data for the lookup. For example:

```
deny senders = cdb;/etc/blocked.senders : \
  mysql;select address from blocked where \
  address='${quote_mysql:$sender_address}'
```

For a single-key lookup type, Exim uses the complete address as the key. Partial matching (section 9.6) cannot be used, and is ignored if specified, with an entry being written to the panic log.

You can configure lookup defaults, as described in section 9.5, but this is useful only for the '*@' type of default. For example, with this lookup:

```
accept senders = lsearch*@;/some/file
```

the file could contains lines like this:

```
user1@domain1.example
*@domain2.example
```

and for the sender address *nimrod@jaeger.example*, the sequence of keys that are tried is:

```
nimrod@jaeger.example
*@jaeger.example
*
```

Warning 1: Do not include a line keyed by '*' in the file, because that would mean that every address matches, thus rendering the test useless.

Warning 2: Do not confuse these two kinds of item:

```
deny recipients = dbm*@;/some/file
deny recipients = *@dbm;/some/file
```

The first does a whole address lookup, with defaulting, as just described, because it starts with a lookup type. The second matches the local part and domain independently, as described in the next paragraph.

- If a pattern contains an @ character, but is not a regular expression and does not begin with a lookup type as described above, 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, the domain is checked in exactly the same way as for a pattern in a domain list. For example, the domain can be wildcarded, refer to a named list, be be or a lookup:

  ```
  deny senders = *@*.spamming.site:\
                 *@+hostile_domains:\
                 bozo@partial-lsearch;/list/of/dodgy/sites:\
                 *@dbm;/bad/domains.db
  ```

  If a local part that begins with an exclamation mark is required, it has to be specified using a regular expression, because otherwise the exclamation mark is treated as a sign of negation.

- If a pattern is not one of the above syntax forms, that is, if a pattern which is not a regular expression or a lookup does not contain an @ character, it is matched against the domain part of the subject address. The only two formats that are recognized this way are a literal domain, or a domain pattern that starts with *. In both these cases, the effect is the same as if *@ preceded the pattern.

Warning: there is an important difference between the address list items in these two examples:

```
senders = +my_list
senders = *@+my_list
```

In the first one, `my_list` is a named address list, whereas in the second example it is a named domain list.

10.14 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 **caseful_local_part** for how Exim deals with this when routing 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 an address list is the string '+caseful', 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, which remains in lower case. However, although independent matches on the domain alone are still performed caselessly, regular expressions that match against an entire address become case-sensitive after '+caseful' has been seen.

10.15 Local part lists

Case-sensitivity in local part lists is handled in the same way as for address lists, as just described. The '+caseful' item can be used if required. In a setting of the **local_parts** option in a router with **caseful_local_part** set false, the subject is lowercased and the matching is initially case-insensitive. In this case, '+caseful' will restore case-sensitive matching in the local part list, but not elsewhere in the router. If **caseful_local_part** is set true in a router, matching in the **local_parts** option is case-sensitive from the start.

If a local part list is indirected to a file (see section 10.3), comments are handled in the same way as address lists – they are recognized only if the # is preceded by white space or the start of the line. Otherwise, local part lists are matched in the same way as domain lists, except that the special items that refer to the local host (`@`, `@[]`, `@mx_any`, `@mx_primary`, and `@mx_secondary`) are not recognized. Refer to section 10.7 for details of the other available item types.

11. String expansions

Many strings in Exim's run time configuration are expanded before use. Some of them are expanded every time they are used; others are expanded only once.

When a string is being expanded it is copied verbatim from left to right 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 in section 11.4 onwards. Backslash is used as an escape character, as described in the following section.

11.1 Literal text in expanded strings

An uninterpreted dollar can be included in an expanded string by putting a backslash in front of it. A backslash can be used to prevent any special character being treated specially in an expansion, including itself. If the string appears in quotes in the configuration file, two backslashes are required because the quotes themselves cause interpretation of backslashes when the string is read in (see section 6.12).

A portion of the string can specified as non-expandable by placing it between two occurrences of `\N`. This is particularly useful for protecting regular expressions, which often contain backslashes and dollar signs. For example:

```
deny senders = \N^\d{8}[a-z]@some\.site\.example$\N
```

On encountering the first `\N`, the expander copies subsequent characters without interpretation until it reaches the next `\N` or the end of the string.

11.2 Character escape sequences in expanded strings

A backslash followed by one of the letters 'n', 'r', or 't' in an expanded string 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, and a backslash followed by 'x' and up to two hexadecimal digits is a hexadecimal encoding.

These escape sequences are also recognized in quoted strings when they are read in. Their interpretation in expansions as well is useful for unquoted strings, and for other cases such as looked-up strings that are then expanded.

11.3 Testing string expansions

Many expansions can be tested by calling Exim with the **-be** option. This takes the command arguments, or lines from the standard input if there are no arguments, runs them through the string expansion code, and writes the results to the standard output. Variables based on configuration values are set up, but since no message is being processed, variables such as **$local_part** have no value. Nevertheless the **-be** option can be useful for checking out file and database lookups, and the use of expansion operators such as **sg**, **substr** and **nhash**.

Exim gives up its root privilege when it is called with the **-be** option, and instead runs under the uid and gid it was called with, to prevent users from using **-be** for reading files to which they do not have access.

11.4 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. **Warning**: Within braces, white space is significant.

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

Substitute the contents of the named variable, for example

```
$local_part
${domain}
```

The second 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 11.8 below. If the name of a non-existent variable is given, the expansion fails.

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

The string is first itself expanded, and then the operation specified by <*op*> is applied to it. For example,

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

The string starts with the first character after the colon, which may be leading white space. A list of operators is given in section 11.5 below. The operator notation is used for simple expansion items that have just one argument, because it reduces the number of braces and therefore makes the string easier to understand.

${extract{<*key*>**}{**<*string1*>**}{**<*string2*>**}{**<*string3*>**}}**

The key and <*string1*> are first expanded separately. The key must not consist entirely of digits. The expanded <*string1*> must be of the form:

<*key1*> = <*value1*> <*key2*> = <*value2*> ...

where the equals signs and spaces (but not both) 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 6.12. The expanded <*string1*> is searched for the value that corresponds to the key. The search is case-insensitive. If the key is found, <*string2*> is expanded, and replaces the whole item; otherwise <*string3*> is used. During the expansion of <*string2*> the variable **$value** contains the value that has been extracted. Afterwards, it is restored to any previous value it might have had.

If {<*string3*>} is omitted, the item is replaced by an empty string if the key is not found. If {<*string2*>} is also omitted, the value that was extracted is used. Thus, for example, these two expansions are identical, and yield ‘2001’:

```
${extract{gid}{uid=1984 gid=2001}}
${extract{gid}{uid=1984 gid=2001}{$value}}
```

Instead of {<*string3*>} the word ‘fail’ (not in curly brackets) can appear, for example:

```
${extract{Z}{A=... B=...}{$value} fail }
```

{<*string2*>} must be present for ‘fail’ to be recognized. When this syntax is used, if the extraction fails, the entire string expansion fails in a way that can be detected by the code in Exim which requested the expansion. This is called ‘forced expansion failure’, and its consequences 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. Such variations are mentioned in the documentation of the option which is expanded.

${extract{<*number*>**}{**<*separators*>**}{**<*string1*>**}{**<*string2*>**}{**<*string3*>**}}**

The <*number*> argument must consist entirely of decimal digits. This is what distinguishes this form of **extract** from the previous kind. It behaves in the same way, except that, instead of extracting a named field, it extracts from <*string1*> the field whose number is given as the first argument. You can use **$value** in <*string2*> or `fail` instead of <*string3*> as before.

The first field is numbered one. If the number is negative, the fields are counted from the end of the string, with the rightmost one numbered -1. If the number given is zero, the entire string is

returned. If the modulus of the number is greater than the number of fields in the string, the result is the expansion of *<string3>*, or the empty string if *<string3>* is not provided. The fields in the string are separated by any one of the characters in the separator string. For example:

```
${extract{2}{:}{x:42:99:& Mailer::/bin/bash}}
```

yields '42', and

```
${extract{-4}{:}{x:42:99:& Mailer::/bin/bash}}
```

yields '99'. Two successive separators mean that the field between them is empty (for example, the fifth field above).

${hash{<*string1*>**}{**<*string2*>**}{**<*string3*>**}}**

This is a textual hashing function, and was the first to be implemented in early versions of Exim. In current releases, there are other hashing functions (numeric, MD5, and SHA-1), which are described below.

The first two strings, after expansion, must be numbers. Call them *<m>* and *<n>*. If you are using fixed values for these numbers, that is, if *<string1>* and *<string2>* do not change when they are expanded, you can use the simpler operator notation that avoids some of the braces:

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

The second number is optional (in both notations).

If *<n>* is greater than or equal to the length of the string, the expansion item 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
```

If *<m>* is not present the value 26 is used, so that only lower case letters appear. For example:

```
${hash{3}{monty}}                 yields  jmg
${hash{5}{monty}}                 yields  monty
${hash{4}{62}{monty python}}      yields  fbWx
```

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

$rheader_<*header name*>**:** or **$rh_**<*header name*>**:**

Substitute the contents of the named message header line, for example

```
$header_reply-to:
```

The newline that terminates a header line is not included in the expansion, but internal newlines (caused by splitting the header line over several physical lines) may be present. The difference between **header** and **rheader** is that the latter inserts the 'raw' contents of header lines without the removal of leading and trailing whitespace.

Header names follow the syntax of RFC 2822, 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.

Only header lines that are common to all copies of a message are visible to this mechanism. These are the original header lines that are received with the message, and any that are added by an ACL **warn** statement or by a system filter. Header lines that are added to a particular copy of a message by a router or transport are not accessible.

For incoming SMTP messages, no header lines are visible in ACLs that are obeyed before the DATA ACL, because the header structure is not set up until the message is received. Header lines that are added by **warn** statements in a RCPT ACL (for example) are saved until the message's incoming header lines are available, at which point they are added. When a DATA ACL is running, however, header lines added by earlier ACLs are visible.

Upper case and lower case letters are synonymous in header names. If the following character is white space, the terminating colon may be omitted, but this is not recommended, because you may then forget it when it is needed. When white space terminates the header name, it 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 11.6 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, up to a maximum length of 64K. A newline character is inserted between each line. For the **header** expansion, for those headers that contain lists of addresses, a comma is also inserted at the junctions between lines. This does not happen for the **rheader** expansion.

${hmac{<*hashname*>**}{**<*secret*>**}{**<*string*>**}}**

This function uses cryptographic hashing (either MD5 or SHA-1) to convert a shared secret and some text into a message authentication code, as specified in RFC 2104. You could produce a similar effect using `${md5:secret_text...}`, but allegedly HMAC provides better defence against deducing the secret.

The hash name must expand to either `md5` or `sha1` at present. For example:

```
${hmac{md5}{somesecret}{$primary_hostname $tod_log}}
```

For the hostname *mail.example.com* and time 2002-10-17 11:30:59, this produces:

```
dd97e3ba5d1a61b5006108f8c8252953
```

As an example of how this might be used, you might put in the main part of an Exim configuration:

```
SPAMSCAN_SECRET=cohgheeLei2thahw
```

In a router or a transport you could then have:

```
headers_add = \
  X-Spam-Scanned: ${primary_hostname} ${message_id} \
  ${hmac{md5}{SPAMSCAN_SECRET}\
  {${primary_hostname},${message_id},$h_message-id:}}
```

Then given a message, you can check where it was scanned by looking at the *X-Spam-Scanned:* header line. If you know the secret, you can check that this header line is authentic by recomputing the authentication code from the host name, message ID and the *Message-id:* header line. This can be done using Exim's **-be** option, or by other means, for example by using the *hmac_md5_hex()* function in Perl.

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

If <*condition*> is true, <*string1*> is expanded and replaces the whole item; otherwise <*string2*> is used. For example,

```
${if eq {$local_part}{postmaster} {yes}{no} }
```

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 is forced to fail if the condition is not true. The available conditions are described in section 11.6 below.

${length{<*string1*>**}{**<*string2*>**}}**

The **length** item is used to extract the initial portion of a string. Both strings are expanded, and the first one must yield a number, <*n*>, say. If you are using a fixed value for the number, that is, if <*string1*> does not change when expanded, you can use the simpler operator notation that avoids some of the braces:

`${length_`<*n*>`:`<*string*>`}`

The result of this item is either the first <*n*> characters or the whole of <*string2*>, whichever is the shorter. Do not confuse **length** with **strlen**, which gives the length of a string.

${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 9. 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 retry or rewrite rule, a routing rule for the **manualroute** router, or any other place where white space is significant, the lookup item must be enclosed in double quotes. The use of data lookups in users' filter files may be locked out by the system administrator.

If the lookup succeeds, <*string1*> is expanded and replaces the entire item. During its expansion, the variable **$value** contains the data returned by the lookup. Afterwards it reverts to the value it had previously (at the outer level it is empty). If the lookup fails, <*string2*> is expanded and replaces the entire item. If {<*string2*>} is omitted, the replacement is null on failure. Alternatively, <*string2*> can itself be a nested lookup, thus providing a mechanism for looking up a default value when the original lookup fails.

If a nested lookup is used as part of <*string1*>, **$value** contains the data for the outer lookup while the parameters of the second lookup are expanded, and also while <*string2*> of the second lookup is expanded, should the second lookup fail.

Instead of {<*string2*>} the word 'fail' can appear, and in this case, if the lookup fails, the entire expansion is forced to fail. If both {<*string1*>} and {<*string2*>} are omitted, the result is the looked up value in the case of a successful lookup, and nothing in the case of failure.

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 9.5 and 9.6 for details).

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.

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, forcing the expansion to fail if it is not found:

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

${nhash}{<*string1*>**}{**<*string2*>**}{**<*string3*>**}}**

The three strings are expanded; the first two must yield numbers. Call them <*n*> and <*m*>. If you are using fixed values for these numbers, that is, if <*string1*> and <*string2*> do not change when they are expanded, you can use the simpler operator notation that avoids some of the braces:

```
${nhash_<n>_<m>:<string>}
```

The second number is optional (in both notations). If there is only one number, the result is a number in the range 0–<*n*>-1. Otherwise, the string is processed by a div/mod hash function that returns two numbers, separated by a slash, in the ranges 0 to <*n*>-1 and 0 to <*m*>-1, respectively. For example,

```
${nhash{8}{64}{supercalifragilisticexpialidocious}}
```

returns the string '6/33'.

${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 additional arguments need be given; the maximum number permitted, including the name of the subroutine, is nine.

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. The return value is a scalar. Whatever you return is evaluated in a scalar context. For example, if you return the name of a Perl vector, the return value is the size of the vector, not its contents.

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 12.

The **redirect** router has an option called **forbid_filter_perl** which locks out the use of this expansion item in filter files.

${readfile{<*file name*>**}{**<*eol string*>**}}**

The file name and end-of-line string are first expanded separately. The file is then read, and its contents replace the entire item. All newline characters in the file are replaced by the end-of-line string if it is present. Otherwise, newlines are left in the string. String expansion is not applied to the contents of the file. If you want this, you must wrap the item in an **expand** operator. If the file cannot be read, the string expansion fails.

The **redirect** router has an option called **forbid_filter_readfile** which locks out the use of this expansion item in filter files.

${readsocket{<*name*>**}{**<*request*>**}{**<*timeout*>**}{**<*eol string*>**}{**<*fail string*>**}}**

This item inserts data that is read from a Unix domain socket into the expanded string. The minimal way of using it uses just two arguments:

```
${readsocket{/socket/name}{request string}}
```

Exim connects to the socket, writes the request string (unless it is an empty string) and reads from the socket until an end-of-file is read. A timeout of 5 seconds is applied. Additional, optional arguments extend what can be done. Firstly, you can vary the timeout. For example:

```
${readsocket{/socket/name}{request-string}{3s}}
```

A fourth argument allows you to change any newlines that are in the data that is read, in the same way as for **readfile** (see above). This example turns them into spaces:

```
${readsocket{/socket/name}{request-string}{3s}{ }}
```

As with all expansions, the substrings are expanded before the processing happens. Errors in these sub-expansions cause the expansion to fail. In addition, the following errors can occur:

- Failure to create a socket file descriptor;
- Failure to connect the socket;
- Failure to write the request-string;
- Timeout on reading from the socket.

By default, any of these errors causes the expansion to fail. However, if you supply a fifth substring, it is expanded and used when any of the above errors occurs. For example:

```
${readsocket{/socket/name}{request-string}{3s}{\n}\
  {socket failure}}
```

You can test for the existence of the socket by wrapping this expansion in `${if exists`, but there is a race condition between that test and the actual opening of the socket, so it is safer to use

the fifth argument if you want to be absolutely sure of avoiding an expansion error for a non-existent socket.

The **redirect** router has an option called **forbid_filter_readsocket** which locks out the use of this expansion item in filter files.

$rheader_<*header name*>**:** or **$rh_**<*header name*>**:**

This item inserts 'raw' header lines. It is described with the **header** expansion item above.

${run{<*command*> <*args*>**}{**<*string1*>**}{**<*string2*>**}}**

The command and its arguments are first expanded separately, and then the command is run in a separate process, but under the same uid and gid. As in other command executions from Exim, a shell is not used by default. If you want a shell, you must explicitly code it. If the command succeeds (gives a zero return code) <*string1*> is expanded and replaces the entire item; during this expansion, the standard output from the command is in the variable **$value**. If the command fails, <*string2*>, if present, is expanded. If it is absent, the result is empty. Alternatively, <*string2*> can be the word 'fail' (not in braces) to force expansion failure if the command does not succeed. If both strings are omitted, the result is the standard output on success, and nothing on failure.

The return code from the command is put in the variable **$runrc**, and this remains set afterwards, so in a filter file you can do things like this:

```
if "${run{x y z}{}}$runrc" is 1 then ...
  elif $runrc is 2 then ...
  ...
endif
```

Warning: In a router or transport, you cannot assume the order in which option values are expanded, except for those pre-conditions whose order of testing is documented. Therefore, you cannot reliably expect to set **$runrc** by the expansion of one option, and use it in another.

The **redirect** router has an option called **forbid_filter_run** which locks out the use of this expansion item in filter files.

${sg{<*subject*>**}{**<*regex*>**}{**<*replacement*>**}}**

This item works like Perl's substitution operator (s) with the global (/g) option; hence its name. However, unlike the Perl equivalent, Exim does not modify the subject string; instead it returns the modified string for insertion into the overall expansion. The item takes three arguments: the subject string, a regular expression, and a substitution string. For example

```
${sg{abcdefabcdef}{abc}{xyz}}
```

yields 'xyzdefxyzdef'. Because all three arguments are expanded before use, if any $ or \ characters are required in the regular expression or in the substitution string, they have to be escaped. For example

```
${sg{abcdef}{^(...)(...)\$}{\$2\$1}}
```

yields 'defabc', and

```
${sg{1=A 4=D 3=C}{\N(\d+)=\N}{K\$1=}}
```

yields 'K1=A K4=D K3=C'. Note the use of `\N` to protect the contents of the regular expression from string expansion.

${substr{<*string1*>**}{**<*string2*>**}{**<*string3*>**}}**

The three strings are expanded; the first two must yield numbers. Call them <*n*> and <*m*>. If you are using fixed values for these numbers, that is, if <*string1*> and <*string2*> do not change when they are expanded, you can use the simpler operator notation that avoids some of the braces:

```
${substr_<n>_<m>:<string>}
```

The second number is optional (in both notations).

The **substr** item can be used to extract more general substrings than **length**. The first number, <*n*>, is a starting offset, and <*m*> is 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 **substr** expansion item can take negative offset values to count from the right-hand 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.

${tr{<*subject*>**}{**<*characters*>**}{**<*replacements*>**}}**

This item does single-character translation on its subject string. The second argument is a list of characters to be translated in the subject string. Each matching character is replaced by the corresponding character from the replacement list. For example

```
${tr{abcdea}{ac}{13}}
```

yields '1b3de1'. If there are duplicates in the second character string, the last occurrence is used. If the third string is shorter than the second, its last character is replicated. However, if it is empty, no translation takes place.

11.5 Expansion operators

For expansion items that perform transformations on a single argument string, the 'operator' notation is used because it is simpler and uses fewer braces. The substring is first expanded before the operation is applied to it. The following operations can be performed:

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

The string is interpreted as an RFC 2822 address, as it might appear in a header line, and the effective address is extracted from it. If the string does not parse successfully, the result is empty.

${base62:<*digits*>**}**

The string must consist entirely of decimal digits. The number is converted to base 62 (sic) and output as a string of six characters, including leading zeros. **Note**: Just to be absolutely clear: this is *not* base64 encoding.

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

The string is interpreted as an RFC 2822 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.

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

This item supports simple arithmetic in expansion strings. The string (after expansion) must be a conventional arithmetic expression, but it is limited to the four basic operators (plus, minus, times, divide) and parentheses. All operations are carried out using integer arithmetic. Plus and minus have a lower priority than times and divide; operators with the same priority are evaluated from left to right. Numbers may be decimal, octal (starting with '0') or hexadecimal (starting with '0x'). A number may be followed by 'K' or 'M' to multiply it by 1024 or 1024∗1024, respectively. Negative numbers are supported. The result of the computation is a decimal representation of the answer (without 'K' or 'M'). For example:

```
${eval:1+1}        yields 2
${eval:1+2*3}      yields 7
${eval:(1+2)*3}    yields 9
```

As a more realistic example, in an ACL you might have

```
deny   message = Too many bad recipients
       condition =                    \
         ${if and {                   \
           {>{$rcpt_count}{10}}       \
           {                          \
           <                          \
             {$recipients_count}      \
             {${eval:$rcpt_count/2}}  \
           }                          \
         }{yes}{no}}
```

The condition is true if there have been more than 10 RCPT commands and fewer than half of them have resulted in a valid recipient.

${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.

${from_utf8:<*string*>**}**

The world is slowly moving towards Unicode, although there are no standards for email yet. However, other applications (including some databases) are starting to store data in Unicode, using UTF-8 encoding. The Unicode code points with values less than 256 are compatible with ASCII and ISO-8859-1 (also known as Latin-1), which of course are single-byte encodings. This operator converts from a UTF-8 string to an ISO-8859-1 string. UTF-8 code values greater than 255 are converted to underscores. The input must be a valid UTF-8 string. If it is not, the result is an undefined sequence of bytes.

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

The **hash** operator is a simpler interface to the hashing function that can be used when the two parameters are fixed numbers (as opposed to strings that change when expanded). The effect is the same as

```
${hash{<n>}{<m>}{<string>}}
```

See the description of the general **hash** item above for details. The abbreviation **h** can be used when **hash** is used as an operator.

${hex2b64:<*hexstring*>**}**

This operator converts a hex string into one that is base64 encoded. This can be useful for processing the output of the MD5 and SHA-1 hashing functions.

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

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

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

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

The **length** operator is a simpler interface to the **length** function that can be used when the parameter is a fixed number (as opposed to a string that changes when expanded). The effect is the same as

```
${length{<number>}{<string>}}
```

See the description of the general **length** item above for details. Note that **length** is not the same as **strlen**. The abbreviation **l** can be used when **length** is used as an operator.

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

The string is interpreted as an RFC 2822 address and the local part is extracted from it. If the string does not parse successfully, the result is empty.

${mask:<*IP address*>**/**<*bit count*>**}**

If the form of the string to be operated on is not an IP address followed by a slash and an integer (that is, a network address in CIDR notation), the expansion fails. Otherwise, this operator converts the IP address to binary, masks off the least significant bits according to the bit count, and converts the result back to text, with mask appended. For example,

```
${mask:10.111.131.206/28}
```

returns the string ‘10.111.131.192/28’. Since this operation is expected to be mostly used for looking up masked addresses in files, the result for an IPv6 address uses dots to separate components instead of colons, because colon terminates a key string in lsearch files. So, for example,

```
${mask:3ffe:ffff:836f:0a00:000a:0800:200a:c031/99}
```

returns the string

```
3ffe.ffff.836f.0a00.000a.0800.2000.0000/99
```

Letters in IPv6 addresses are always output in lower case.

${md5:<*string*>**}**

The **md5** operator computes the MD5 hash value of the string, and returns it as a 32-digit hexadecimal number, in which any letters are in lower case.

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

The **nhash** operator is a simpler interface to the numeric hashing function that can be used when the two parameters are fixed numbers (as opposed to strings that change when expanded). The effect is the same as

```
${nhash{<n>}{<m>}{<string>}}
```

See the description of the general **nhash** item above for details.

${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. Newlines and carriage returns are converted to `\n` and `\r`, respectively 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. In particular, if you are creating a new email address from the contents of **$local_part** (or any other unknown data), you should always use this operator.

${quote_<*lookup-type*>**:**<*string*>**}**

This operator applies lookup-specific quoting rules to the string. Each query-style lookup type has its own quoting rules which are described with the lookups in chapter 9. For example,

```
${quote_ldap:two + two}
```

returns ‘two%20%5C+%20two’. For single-key lookup types, no quoting is necessary and this operator yields an unchanged string.

${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.

${rfc2047:<*string*>**}**

This operator encodes text according to the rules of RFC 2047. This is an encoding that is used in header lines to encode special characters. It is assumed that the input string is in ISO-8859-1 encoding. If the string contains only characters in the range 33–126, and no instances of the characters

```
? = ( ) < > @ , ; : \ " . [ ] _
```

it is not modified. Otherwise, the result is the RFC 2047 encoding, as a single ‘coded word’.

${sha1:<*string*>**}**

The **sha1** operator computes the SHA-1 hash value of the string, and returns it as a 40-digit hexadecimal number, in which any letters are in upper case.

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

The string, after expansion, must be a file path. A call to the *stat()* function is made for this path. If *stat()* fails, an error occurs and the expansion fails. If it succeeds, the data from the stat replaces the item, as a series of <*name*>=<*value*> pairs, where the values are all numerical, except for the value of ‘smode’. The names are: ‘mode’ (giving the mode as a 4-digit octal number), ‘smode’ (giving the mode in symbolic format as a 10-character string, as for the *ls* command), ‘inode’, ‘device’, ‘links’, ‘uid’, ‘gid’, ‘size’, ‘atime’, ‘mtime’, and ‘ctime’. You can extract individual fields using the **extract** expansion item. **Warning**: The file size may be incorrect on 32-bit systems for files larger than 2GB.

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

The item is replace by the length of the expanded string, expressed as a decimal number. **Note**: Do not confuse **strlen** with **length**.

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

The **substr** operator is a simpler interface to the **substr** function that can be used when the two parameters are fixed numbers (as opposed to strings that change when expanded). The effect is the same as

```
${substr{<start>}{<length>}{<string>}}
```

See the description of the general **substr** item above for details. The abbreviation **s** can be used when **substr** is used as an operator.

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

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

11.6 Expansion conditions

The following conditions are available for testing by the **${if** construct while expanding strings:

!<*condition*>

Preceding any condition with an exclamation mark 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
```

For example,

```
${if >{$message_size}{10M} ...
```

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 upper or lower case), signifying multiplication by 1024 or 1024∗1024, respectively.

crypteq {<*string1*>**}{**<*string2*>**}**

This condition is included in the Exim binary if it is built to support any authentication mechanisms (see chapter 32). Otherwise, it is necessary to define SUPPORT_CRYPTEQ in **Local/Makefile** to get **crypteq** included in the binary.

The **crypteq** condition has two arguments. The first is encrypted and compared against the second, which is already encrypted. The second string may be in the LDAP form for storing encrypted strings, which starts with the encryption type in curly brackets, followed by the data. If the second string does not begin with ‘{’ it is assumed to be encrypted with *crypt()* or *crypt16()* (see below), since such strings cannot begin with ‘{’. Typically this will be a field from a password file.

An example of an encrypted string in LDAP form is:

```
{md5}CY9rzUYh03PK3k6DJie09g==
```

If such a string appears directly in an expansion, the curly brackets have to be quoted, because they are part of the expansion syntax. For example:

```
${if crypteq {test}{\{md5\}CY9rzUYh03PK3k6DJie09g==}{yes}{no}}
```

The following encryption types are supported:

- **{md5}** computes the MD5 digest of the first string, and expresses this as printable characters to compare with the remainder of the second string. If the length of the comparison string is 24, Exim assumes that it is base64 encoded (as in the above example). If the length is 32,

Exim assumes that it is a hexadecimal encoding of the MD5 digest. If the length not 24 or 32, the comparison fails.

- **{sha1}** computes the SHA-1 digest of the first string, and expresses this as printable characters to compare with the remainder of the second string. If the length of the comparison string is 28, Exim assumes that it is base64 encoded. If the length is 40, Exim assumes that it is a hexadecimal encoding of the SHA-1 digest. If the length is not 28 or 40, the comparison fails.
- **{crypt}** calls the *crypt()* function, which uses only the first eight characters of the password.
- **{crypt16}** calls the *crypt16()* function (also known as *bigcrypt()*), which uses up to 16 characters of the password.

Exim has its own version of *crypt16()* (which is just a double call to *crypt()*). For operating systems that have their own version, setting HAVE_CRYPT16 in **Local/Makefile** when building Exim causes it to use the operating system version instead of its own. This option is set by default in the OS-dependent **Makefile** for those operating systems that are known to support *crypt16()*.

If you do not put any curly bracket encryption type in a **crypteq** comparison, the default is either `{crypt}` or `{crypt16}`, as determined by the setting of DEFAULT_CRYPT in **Local/Makefile**. The default default is `{crypt}`. Whatever the default, you can always use either function by specifying it explicitly in curly brackets.

Note that if a password is no longer than 8 characters, the results of encrypting it with *crypt()* and *crypt16()* are identical. That means that *crypt16()* is backwards compatible, as long as nobody feeds it a password longer than 8 characters.

def:<*variable name*>

The **def** condition must be followed by the name of one of the expansion variables defined in section 11.8. 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.

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. Use the **lc** or **uc** expansion operators to force both strings to the same case if you want to do a caseless comparison.

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. The use of the **exists** test in users' filter files may be locked out by the system administrator.

first_delivery

This condition, which has no data, is true during a message's first delivery attempt. It is false during any subsequent delivery attempts.

ldapauth {<*ldap query*>**}**

This condition supports user authentication using LDAP. See section 9.10 for details of how to use LDAP in lookups and the syntax of queries. For this use, the query must contain a user name and password. The query itself is not used, and can be empty. The condition is true if the password is not empty, and the user name and password are accepted by the LDAP server. An empty password is rejected without calling LDAP because LDAP binds with an empty password are considered anonymous regardless of the username, and will succeed in most configurations. See chapter 32 for details of SMTP authentication, and chapter 33 for an example of how this can be used.

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. 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*>. The easiest approach is to use the `\N` feature to disable expansion of the regular expression. For example,

```
${if match {$local_part}{\N^\d{3}\N} ...
```

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.

pam {<*string1*>**:**<*string2*>**:...}**

Pluggable Authentication Modules (**http://www.kernel.org/pub/linux/libs/pam/**) are a facility which is available in the latest releases of Solaris and in some GNU/Linux distributions. The Exim support, which is intended for use in conjunction with the SMTP AUTH command, is available only if Exim is compiled with

```
SUPPORT_PAM=yes
```

in **Local/Makefile**. You probably need to add **-lpam** to EXTRALIBS, and in some releases of GNU/Linux **-ldl** is also needed.

The argument string is first expanded, and the result must be a colon-separated list of strings. The PAM module is initialized with the service name 'exim' and the user name taken from the first item in the colon-separated data string (<*string1*>). The remaining items in the data string are passed over in response to requests from the authentication function. In the simple case there will only be one request, for a password, so the data consists of just two strings.

There can be problems if any of the strings are permitted to contain colon characters. In the usual way, these have to be doubled to avoid being taken as separators. If the data is being inserted from a variable, the **sg** expansion item can be used to double any existing colons. For example, the configuration of a LOGIN authenticator might contain this setting:

```
server_condition = ${if pam{$1:${sg{$2}{:}{::}}}{yes}{no}}
```

For a PLAIN authenticator you could use:

```
server_condition = ${if pam{$2:${sg{$3}{:}{::}}}{yes}{no}}
```

In some operating systems, PAM authentication can be done only from a process running as root. Since Exim is running as the Exim user when receiving messages, this means that PAM cannot be used directly in those systems. A patched version of the *pam_unix* module that comes with the Linux PAM package is available from **http://www.e-admin.de/pam_exim/**. The patched module allows one special uid/gid combination, in addition to root, to authenticate. If you build the patched module to allow the Exim user and group, PAM can then be used from an Exim authenticator.

pwcheck {<*string1*>**:**<*string2*>**}**

This condition supports user authentication using the Cyrus *pwcheck* daemon. This is one way of making it possible for passwords to be checked by a process that is not running as root.

The pwcheck support is not included in Exim by default. You need to specify the location of the pwcheck daemon's socket in **Local/Makefile** before building Exim. For example:

```
CYRUS_PWCHECK_SOCKET=/var/pwcheck/pwcheck
```

You do not need to install the full Cyrus software suite in order to use the pwcheck daemon. You can compile and install just the daemon alone from the Cyrus SASL library. Ensure that *exim* is the only user that has access to the **/var/pwcheck** directory.

The **pwcheck** condition takes one argument, which must be the user name and password, separated by a colon. For example, in a LOGIN authenticator configuration, you might have this:

```
server_condition = ${if pwcheck{$1:$2}{1}{0}}
```

queue_running

This condition, which has no data, is true during delivery attempts that are initiated by queue runner processes, and false otherwise.

radius {<*authentication string*>**}**

Radius authentication (RFC 2865) is supported in a similar way to PAM. You must set RADIUS_CONFIG_FILE in **Local/Makefile** to specify the location of the Radius client configuration file in order to build Exim with Radius support. You may also have to supply a suitable setting in EXTRALIBS so that the Radius library can be found when Exim is linked. The string specified by RADIUS_CONFIG_FILE is expanded and passed to the Radius client library, which calls the Radius server. The condition is true if the authentication is successful. For example

```
server_condition = ${if radius{<arguments>}{yes}{no}}
```

11.7 Combining expansion conditions

Several conditions can be tested at once by combining them using the **and** and **or** combination conditions. Note that **and** and **or** are complete conditions on their own, and precede their lists of sub-conditions. Each sub-condition must be enclosed in braces within the overall braces that contain the list. No repetition of **if** is used.

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. For example,

```
${if or {{eq{$local_part}{spqr}}{eq{$domain}{testing.com}}}...
```

When a true sub-condition is found, the following ones are parsed but not evaluated. If there are several 'match' sub-conditions the values of the numeric variables afterwards 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. If there are several 'match' sub-conditions, the values of the numeric variables afterwards are taken from the last one. When a false sub-condition is found, the following ones are parsed but not evaluated.

11.8 Expansion variables

The variables that are available for use in expansion strings are:

$0, **$1**, etc: When a **match** 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.

$acl_c0 – **$acl_c9**: Values can be placed in these variables by the **set** modifier in an ACL. The values persist throughout the lifetime of an SMTP connection. They can be used to pass information between ACLs and different invocations of the same ACL. As their name suggests, these variables are set only during ACL processing. At all other times (in particular, during message delivery) they are empty.

$acl_m0 – **$acl_m9**: Values can be placed in these variables by the **set** modifier in an ACL. They retain their values while a message is being received, including during the running of a *local_scan()* function, but are reset afterwards. They are also reset by MAIL, RSET, EHLO, HELO, and after starting a TLS session.

$acl_verify_message: During the expansion of the **message** modifier in an ACL statement after an address verification has failed, this variable contains the original failure message that will be overridden by the expanded string.

$address_data: This variable is set by means of the **address_data** option in routers. The value then remains with the address while it is processed by subsequent routers and eventually a transport. If the transport is handling multiple addresses, the value from the first address is used. See chapter 14 for more details. **Note**: the contents of **$address_data** are visible in user filter files.

If **$address_data** is set when the routers are called to verify an address from an ACL, the final value remains available in subsequent conditions in the ACL statement. If routing the address caused it to be redirected to a single address, the child address is also routed as part of the verification, and in this case the final value of **$address_data** is from the child's routing.

$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.

$authenticated_id: When a server successfully authenticates a client it may be configured to preserve some of the authentication information in the variable **$authenticated_id** (see chapter 32). For example, a user/password authenticator configuration might preserve the user name for use in the routers. When a message is submitted locally (that is, not over a TCP connection), the value of **$authenticated_id** is the login name of the calling process.

$authenticated_sender: When a client host has authenticated itself, Exim pays attention to the AUTH= parameter on the SMTP MAIL command, provided the setting of **server_mail_auth_condition** (see chapter 32) permits it. Otherwise, it accepts the syntax, but ignores the data. Unless the data is the

string '<>', it is set as the authenticated sender of the message, and the value is available during delivery in the **$authenticated_sender** variable. When a message is submitted locally (that is, not over a TCP connection), the value of **$authenticated_sender** is an address constructed from the login name of the calling process and **$qualify_domain**.

$authentication_failed: This variable is set to '1' in an Exim server if a client issues an AUTH command that does not succeed. Otherwise it is set to '0'. This makes it possible to distinguish between 'did not try to authenticate' (**$sender_host_authenticated** is empty and **$authentication_failed** is set to '0') and 'tried to authenticate but failed' (**$sender_host_authenticated** is empty and **$authentication_failed** is set to '1'). Failure includes any negative response to an AUTH command, including (for example) an attempt to use an undefined mechanism.

$body_linecount: When a message is being received or delivered, this variable contains the number of lines in the message's body.

$bounce_recipient: This is set to the recipient address of a bounce message while Exim is creating it. It is useful if a customized bounce message text file is in use (see chapter 40).

$caller_gid: The group id under which the process that called Exim was running. This is not the same as the group id of the originator of a message (see **$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.

$dnslist_domain: When a client host is found to be on a DNS (black) list, the list's domain name is put into this variable so that it can be included in the rejection message.

$dnslist_text: When a client host is found to be on a DNS (black) list, the contents of any associated TXT record are placed in this variable.

$dnslist_value: When a client host is found to be on a DNS (black) list, the IP address from the resource record is placed in this variable.

$domain: When an address is being routed, or delivered on its own, this variable contains the domain. Global address rewriting happens when a message is received, so the value of **$domain** during routing and delivery is the value after rewriting. **$domain** is set during user filtering, but not during system filtering, because a message may have many recipients and the system filter is called just once.

When more than one address is being delivered at once (for example, several RCPT commands in one SMTP delivery), **$domain** is set only if they all have the same domain. Transports can be restricted to handling only one domain at a time if the value of **$domain** 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 22.

At the end of a delivery, if all deferred addresses have the same domain, it is set in **$domain** during the expansion of **delay_warning_condition**.

The **$domain** variable is also used in some other circumstances:

- When an ACL is running for a RCPT command, **$domain** contains the domain of the recipient address.
- When a rewrite item is being processed (see chapter 30), **$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.

- Whenever a domain list is being scanned, **$domain** contains the subject domain.
- When the **smtp_etrn_command** option is being expanded, **$domain** contains the complete argument of the ETRN command (see section 42.9).

$domain_data: When the **domains** option on a router matches a domain by means of a lookup, the data read by the lookup is available during the running of the router as **$domain_data**. In addition, if the driver routes the address to a transport, the value is available in that transport. If the transport is handling multiple addresses, the value from the first address is used.

$domain_data is also set when the **domains** condition in an ACL matches a domain by means of a lookup. The data read by the lookup is available during the rest of the ACL statement. In all other situations, this variable expands to nothing.

$header_<*name*>: This is not strictly an expansion variable. It is expansion syntax for inserting the message header line with the given name. Note that the name must be terminated by colon or white space, because it may contain a wide variety of characters. Note also that braces must *not* be used.

$home: When the **check_local_user** option is set for a router, the user's home directory is placed in **$home** when the check succeeds. In particular, this means it is set during the running of users' filter files. A router may also explicitly set a home directory for use by a transport; this can be overridden by a setting on the transport itself.

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

$host: When the **smtp** transport is expanding its options for encryption using TLS, **$host** contains the name of the host to which it is connected. Likewise, when used in the client part of an authenticator configuration (see chapter 32), **$host** contains the name of the server to which the client is connected. When used in a transport filter (see chapter 23) **$host** refers to the host involved in the current connection. When a local transport is run as a result of a router that sets up a host list, **$host** contains the name of the first host.

$host_address: This variable is set to the remote host's IP address whenever **$host** is set for a remote connection.

$host_data: If a **hosts** condition in an ACL is satisfied by means of a lookup, the result of the lookup is made available in the **$host_data** variable. This allows you, for example, to do things like this:

```
deny  hosts = net-lsearch;/some/file
      message = $host_data
```

$host_lookup_failed: This variable contains '1' if the message came from a remote host and there was an attempt to look up the host's name from its IP address, but the attempt failed. Otherwise the value of the variable is '0'.

$inode: The only time this variable is set is while expanding the **directory_file** option in the **appendfile** transport. The variable contains the inode number of the temporary file which is about to be renamed. It can be used to construct a unique name for the file.

$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.

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

$ldap_dn: This variable, which is available only when Exim is compiled with LDAP support, contains the DN from the last entry in the most recently successful LDAP lookup.

$load_average: This variable contains the system load average, multiplied by 1000 to that it is an integer. For example, if the load average is 0.21, the value of the variable is 210. The value is recomputed every time the variable is referenced.

$local_part: When an address is being routed, or delivered on its own, this variable contains the local part. When a number of addresses are being delivered together (for example, multiple RCPT commands in an SMTP session), **$local_part** is not set.

Global address rewriting happens when a message is received, so the value of **$local_part** during routing and delivery is the value after rewriting. **$local_part** is set during user filtering, but not during system filtering, because a message may have many recipients and the system filter is called just once.

If a local part prefix or suffix has been recognized, it is not included in the value of **$local_part** during routing and subsequent delivery. The values of any prefix or suffix are in **$local_part_prefix** and **$local_part_suffix**, respectively.

When a message is being delivered to a file, pipe, or autoreply transport as a result of aliasing or forwarding, **$local_part** is set to the local part of the parent address, not to the file name or command (see **$address_file** and **$address_pipe**).

When an ACL is running for a RCPT command, **$local_part** contains the local part of the recipient address.

When a rewrite item is being processed (see chapter 30), **$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.

In all cases, all quoting is removed from the local part. For example, for both the addresses

```
"abc:xyz"@test.example
abc\:xyz@test.example
```

the value of **$local_part** is

```
abc:xyz
```

If you use **$local_part** to create another address, you should always wrap it inside a quoting operator. For example, in a **redirect** router you could have:

```
data = ${quote:$local_part}@new.domain.example
```

$local_part_data: When the **local_parts** option on a router matches a local part by means of a lookup, the data read by the lookup is available during the running of the router as **$local_part_data**. In addition, if the driver routes the address to a transport, the value is available in that transport. If the transport is handling multiple addresses, the value from the first address is used.

$local_part_data is also set when the **local_parts** condition in an ACL matches a local part by means of a lookup. The data read by the lookup is available during the rest of the ACL statement. In all other situations, this variable expands to nothing.

$local_part_prefix: When an address is being routed or delivered, and a specific prefix for the local part was recognized, it is available in this variable, having been removed from **$local_part**.

$local_part_suffix: When an address is being routed or delivered, and a specific suffix for the local part was recognized, it is available in this variable, having been removed from **$local_part**.

$local_scan_data: This variable contains the text returned by the *local_scan()* function when a message is received. See chapter 38 for more details.

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

$message_age: This variable is set at the start of a delivery attempt to contain the number of seconds since the message was received. It does not change during a single delivery attempt.

$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 put into the variable 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. Binary zeros are also converted into spaces.

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

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

$message_headers: This variable contains a concatenation of all the header lines when a message is being processed, except for lines added by routers or transports. The header lines are separated by newline characters.

$message_id: When a message is being received or delivered, this variable contains the unique message id that is used by Exim to identify the message. An id is not allocated to a message until after its header has been successfully received.

$message_size: When a message is being processed, this variable contains its size in bytes. In most cases, the size includes those headers that were received with the message, but not those (such as *Envelope-to:*) that are added to individual deliveries as they are written. However, there is one special case: during the expansion of the **maildir_tag** option in the **appendfile** transport while doing a delivery in maildir format, the value of **$message_size** is the precise size of the file that has been written. See also **$message_body_size** and **$body_linecount**.

While running an ACL at the time of an SMTP RCPT command, **$message_size** contains the size supplied on the MAIL command, or zero if no size was given. The value may not, of course, be truthful.

$n0 – **$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 **$parent_domain** only when there is more than one level of aliasing or forwarding. When more than one address is being delivered in a single transport run, **$original_domain** is not set.

If new an address is created by means of a **deliver** command in a system filter, it is set up with an artificial 'parent' address. This has the local part *system-filter* and the default qualify domain.

$original_local_part: When a top-level address is being processed for delivery, this contains the same value as **$local_part**, unless a prefix or suffix was removed from the local part, in which case **$original_local_part** contains the full local part. When a 'child' address (for example, generated by an alias, forward, or filter file) is being processed, this variable contains the full local part of the original address. If the router that did the redirection processed the local part case-insensitively, the value in **$original_local_part** is in lower case. This variable differs from **$parent_local_part** only when there is more than one level of aliasing or forwarding. When more than one address is being delivered in a single transport run, **$original_local_part** is not set.

If new an address is created by means of a **deliver** command in a system filter, it is set up with an artificial 'parent' address. This has the local part *system-filter* and the default qualify domain.

$originator_gid: The value of **$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.

$parent_domain: This variable is similar to **$original_domain** (see above), except that it refers to the immediately preceding parent address.

$parent_local_part: This variable is similar to **$original_local_part** (see above), except that it refers to the immediately preceding parent address.

$pid: This variable contains the current process id.

$pipe_addresses: This is not an expansion variable, but is mentioned here because the string '$pipe_addresses' is handled specially in the command specification for the **pipe** transport (chapter 28) and in transport filters (described under **transport_filter** in chapter 23). 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. If *uname()* returns a single-component name, Exim calls *gethostbyname()* (or *getipnodebyname()* where available) in an attempt to acquire a fully qualified host name.

$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**.

$rcpt_count: When a message is being received by SMTP, this variable contains the number of RCPT commands received for the current message. If this variable is used in a RCPT ACL, its value includes the current command.

$rcpt_defer_count: When a message is being received by SMTP, this variable contains the number of RCPT commands in the current message that have previously been rejected with a temporary (4*xx*) response.

$rcpt_fail_count: When a message is being received by SMTP, this variable contains the number of RCPT commands in the current message that have previously been rejected with a permanent (5*xx*) response.

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

$received_protocol: When a message is being processed, this variable contains the name of the protocol by which it was received. See also the **-oMr** option.

$recipients: This variable contains a list of envelope recipients for a message. A comma and a space separate the addresses in the replacement text. However, the variable is not generally available, to prevent exposure of Bcc recipients in unprivileged users' filter files. You can use **$recipients** only

(1) In a system filter file.

(2) In the DATA or non-SMTP ACL, that is, in the final ACL for accepting a message.

$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. While a message is being received over SMTP, the number increases for each accepted recipient. It can be referenced in an ACL.

$reply_address: When a message is being processed, this variable contains the contents of the *Reply-To:* header line if one exists and it is not empty, or otherwise the contents of the *From:* header line.

$return_path: When a message is being delivered, this variable contains the return path – the sender field that will be sent as part of the envelope. It is not enclosed in <> characters. In many cases, **$return_path** has the same value as **$sender_address**, but if, for example, an incoming message to a mailing list has been expanded by a router which specifies a different address for bounce messages, **$return_path** contains the new bounce address, whereas **$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 40).

$runrc: This variable contains the return code from a command that is run by the **${run...}** expansion item. **Warning**: In a router or transport, you cannot assume the order in which option values are

expanded, except for those pre-conditions whose order of testing is documented. Therefore, you cannot reliably expect to set **$runrc** by the expansion of one option, and use it in another.

$self_hostname: When an address is routed to a supposedly remote host that turns out to be the local host, what happens is controlled by the **self** generic router option. One of its values causes the address to be passed to another router. When this happens, **$self_hostname** is set to the name of the local host that the original 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 bounce messages, 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 is received from a remote host, this variable contains the host name and IP address in a single string. It ends with the IP address in square brackets, followed by a colon and a port number if the logging of ports is enabled. 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 is received from a remote host that has issued a HELO or EHLO command, 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 is received from a remote host, this variable contains that host's IP address. For locally submitted messages, it is empty.

$sender_host_authenticated: This variable contains the name (not the public name) of the authenticator driver which successfully authenticated the client from which the message was received. It is empty if there was no successful authentication.

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

Exim does not always look up every calling host's name. If you want maximum efficiency, you should arrange your configuration so that it avoids these lookups altogether. The lookup happens only if any of the following are true:

(1) The calling host matches the list in **host_lookup**. The default for this option is ∗, so it must be changed if any lookups are to be avoided.

(2) Exim needs the host name in order to test an item in a host list. The items that require this are described in section 10.11. A common mistake is to forget to use 'net-' before a query-style lookup that actually looks up the host address.

(3) The calling host matches **helo_try_verify_hosts** or **helo_verify_hosts**. In this case, the host name is required to compare with the name quoted in any EHLO or HELO commands that the client issues.

(4) The remote host issues a EHLO or HELO command that quotes one of the domains in **helo_lookup_domains**. The default value of this option is

```
helo_lookup_domains = @ : @[]
```

which causes a lookup if a remote host (incorrectly) gives the server's name or IP address in an EHLO or HELO command.

$sender_host_port: When a message is received from a remote host, this variable contains the port number that was used on the remote host.

$sender_ident: When a message is 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, followed by a colon and a port number if port logging is enabled. When the first item is an IP address, the port is recorded as 'port=*xxxx*' inside the parentheses.

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.

$smtp_command_argument: While an ACL is running to check an AUTH, EHLO, EXPN, ETRN, HELO, or VRFY command, this variable contains the argument for the SMTP command.

$sn0 – **$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.

$tls_certificate_verified: This variable is set to '1' if a TLS certificate was verified when the message was received, and '0' otherwise.

$tls_cipher: When a message is received from a remote host over an encrypted SMTP connection, this variable is set to the cipher suite that was negotiated, for example DES-CBC3-SHA. In other circumstances, in particular, for message received over unencrypted connections, the variable is empty. See chapter 36 for details of TLS support.

$tls_peerdn: When a message is received from a remote host over an encrypted SMTP connection, and Exim is configured to request and verify a certificate from the client, the value of the Distinguished Name of the certificate is made available in the **$tls_peerdn** during subsequent processing.

$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_epoch: The time and date as a number of seconds since the start of the Unix epoch.

$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 UTC, with positive values used for timezones that are ahead (east) of UTC, and negative values for those that are behind (west).

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

$tod_logfile: This variable contains the date in the format yyyymmdd. This is the format that is used for datestamping log files when **log_file_path** contains the `%D` flag.

$tod_zone: This variable contains the numerical value of the local timezone, for example: -0500.

$tod_zulu: This variable contains the UTC date and time in 'Zulu' format, as specified by ISO 8601, for example: 20030221154023Z.

$value: This variable contains the result of an expansion lookup, extraction operation, or external command, as described above.

$version_number: The version number of Exim.

$warn_message_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 40.2.

$warn_message_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 40.2.

12. 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. Also, note that this does not mean that Exim is necessarily running as root when Perl is called at a later time. 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 Perl subroutine is evaluated in a scalar context before it is passed back to Exim to be inserted into the expanded string. If the return value is *undef*, 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, 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.

13. Main configuration

The first part of the run time configuration file contains three types of item:

- Macro definitions: These lines start with an upper case letter. See section 6.4 for details of macro processing.
- Named list definitions: These lines start with one of the words ‘domainlist’, ‘hostlist’, ‘addresslist’, or ‘localpartlist’. Their use is described in section 10.5.
- Main configuration settings: Each setting occupies one line of the file (with possible continuations). If any setting is preceded by the word ‘hide’, the **-bP** command line option displays its value to admin users only. See section 6.6 for a description of the syntax of these option settings.

This chapter specifies all the main configuration options, along with their types and default values. For ease of finding a particular option, they appear in alphabetical order in section 13.22 below. However, because there are now so many options, they are first listed briefly in functional groups, as an aid to finding the name of the option you are looking for.

13.1 Miscellaneous

bi_command	to run for **-bi** command line option
keep_malformed	for broken files – should not happen
localhost_number	for unique message ids in clusters
message_logs	keep per-message logs
message_body_visible	how much to show in **$message_body**
print_topbitchars	top-bit characters are printing
split_spool_directory	use multiple directories
timezone	force time zone

13.2 Exim parameters

exim_group	override compiled-in value
exim_path	override compiled-in value
exim_user	override compiled-in value
primary_hostname	default from *uname()*
spool_directory	override compiled-in value

13.3 Privilege controls

admin_groups	groups that are Exim admin users
deliver_drop_privilege	drop root for delivery processes
local_from_check	insert *Sender:* if necessary
local_from_prefix	for testing *From:* for local sender
local_from_suffix	for testing *From:* for local sender
never_users	do not run deliveries as these
prod_requires_admin	forced delivery requires admin user
queue_list_requires_admin	queue listing requires admin user
trusted_groups	groups that are trusted
trusted_users	users that are trusted

13.4 Logging

log_file_path	override compiled-in value
log_selector	set/unset optional logging
log_timezone	add timezone to log lines

preserve_message_logs	in another directory
syslog_facility	set syslog ‘facility’ field
syslog_processname	set syslog ‘ident’ field
syslog_timestamp	timestamp syslog lines

13.5 Frozen messages

auto_thaw	sets time for retrying frozen messages
freeze_tell	send message when freezing
move_frozen_messages	to another directory
timeout_frozen_after	keep frozen messages only so long

13.6 Data lookups

ldap_default_servers	used if no server in query
ldap_version	set protocol version
lookup_open_max	lookup files held open
mysql_servers	as it says
oracle_servers	as it says
pgsql_servers	as it says

13.7 Message ids

message_id_header_domain	used to build *Message-ID:* header
message_id_header_text	ditto

13.8 Embedded Perl Startup

perl_at_start	always start the interpreter
perl_startup	code to obey when starting Perl

13.9 Daemon

daemon_smtp_port	default port
local_interfaces	on which to listen, with optional ports
pid_file_path	override compiled-in value

13.10 Resource control

check_log_inodes	before accepting a message
check_log_space	before accepting a message
check_spool_inodes	before accepting a message
check_spool_space	before accepting a message
deliver_queue_load_max	no queue deliveries if load high
smtp_load_reserve	SMTP from reserved hosts if load high
queue_only_load	queue incoming if load high

13.11 Policy controls

acl_not_smtp	set ACL for non-SMTP messages
acl_smtp_auth	set ACL for AUTH
acl_smtp_connect	set ACL for connection
acl_smtp_data	set ACL for DATA
acl_smtp_etrn	set ACL for ETRN
acl_smtp_expn	set ACL for EXPN
acl_smtp_helo	set ACL for EHLO or HELO

acl_smtp_mail	set ACL for MAIL
acl_smtp_rcpt	set ACL for RCPT
acl_smtp_starttls	set ACL for STARTTLS
acl_smtp_vrfy	set ACL for VRFY
header_maxsize	total size of message header
header_line_maxsize	individual header line limit
helo_verify_hosts	HELO checked for these hosts
host_lookup	host name looked up for these hosts
host_reject_connection	reject connection from these hosts
hosts_treat_as_local	useful in some cluster configurations
local_scan_timeout	timeout for *local_scan()*
message_size_limit	for all messages
percent_hack_domains	recognize %-hack for these domains

13.12 Callout cache

callout_domain_negative_expire	timeout for negative domain cache item
callout_domain_positive_expire	timeout for positive domain cache item
callout_negative_expire	timeout for negative address cache item
callout_positive_expire	timeout for positive address cache item
callout_random_local_part	string to use for ‘random’ testing

13.13 TLS

tls_advertise_hosts	advertise TLS to these hosts
tls_certificate	location of server certificate
tls_dhparam	DH parameters for server
tls_privatekey	location of server private key
tls_try_verify_hosts	try to verify client certificate
tle_verify_certificates	expected client certificates
tls_verify_hosts	insist on client certificate verify

13.14 Local user handling

finduser_retries	useful in NIS environments
gecos_name	used when creating *Sender:*
gecos_pattern	ditto
max_username_length	for systems that truncate
unknown_login	used when no login name found
unknown_username	ditto
uucp_from_pattern	for recognizing ‘From ’ lines
uucp_from_sender	ditto

13.15 Incoming messages

header_maxsize	total size of message header
header_line_maxsize	individual header line limit
percent_hack_domains	recognize %-hack for these domains
receive_timeout	for non-SMTP messages
received_header_text	expanded to make *Received:*
received_headers_max	for mail loop detection
recipient_unqualified_hosts	may send unqualified recipients
recipients_max	limit per message
recipients_max_reject	permantely reject excess

13.16 Incoming SMTP

rfc1413_hosts	make ident calls to these hosts
rfc1413_query_timeout	zero disables ident calls
sender_unqualified_hosts	may send unqualified senders
smtp_accept_keepalive	some TCP/IP magic
smtp_accept_max	simultaneous incoming connections
smtp_accept_max_nommail	non-mail commands
smtp_accept_max_nonmail_hosts	hosts to which the limit applies
smtp_accept_max_per_connection	messages per connection
smtp_accept_max_per_host	connections from one host
smtp_accept_queue	queue mail if more connections
smtp_accept_queue_per_connection	queue if more messages per connection
smtp_accept_reserve	only reserve hosts if more connections
smtp_banner	text for welcome banner
smtp_check_spool_space	from SIZE on MAIL command
smtp_connect_backlog	passed to TCP/IP stack
smtp_enforce_sync	of SMTP command/responses
smtp_etrn_command	what to run for ETRN
smtp_etrn_serialize	only one at once
smtp_load_reserve	only reserve hosts if this load
smtp_max_unknown_commands	before dropping connection
smtp_ratelimit_hosts	apply ratelimiting to these hosts
smtp_ratelimit_mail	ratelimit for MAIL commands
smtp_ratelimit_rcpt	ratelimit for RCPT commands
smtp_receive_timeout	per command or data line
smtp_reserve_hosts	these are the reserve hosts
smtp_return_error_details	give detail on rejections

13.17 SMTP extensions

accept_8bitmime	advertise 8BITMIME
auth_advertise_hosts	advertise AUTH to these hosts
ignore_fromline_hosts	allow ‘From ’ from these hosts
ignore_fromline_local	allow ‘From ’ from local SMTP
pipelining_advertise_hosts	advertise pipelining to these hosts
tls_advertise_hosts	advertise TLS to these hosts

13.18 Processing messages

allow_domain_literals	recognize domain literal syntax
allow_mx_to_ip	allow MX to point to IP address
allow_utf8_domains	in addresses
delivery_date_remove	from incoming messages
drop_cr	from local incoming messages
envelope_to_remote	from incoming messages
extract_addresses_remove_arguments	affects **-t** processing
qualify_domain	default for senders
qualify_recipient	default for recipients
return_path_remove	from incoming messages
strip_excess_angle_brackets	in addresses
strip_trailing_dot	at end of addresses
untrusted_set_sender	untrusted can set envelope sender

13.19 System filter

system_filter	locate system filter
system_filter_directory_transport	transport for delivery to a directory
system_filter_file_transport	transport for delivery to a file
system_filter_group	group for filter running
system_filter_pipe_transport	transport for delivery to a pipe
system_filter_reply_transport	transport for autoreply delivery
system_filter_user	user for filter running

13.20 Routing and delivery

dns_again_means_nonexist	for broken domains
dns_check_names_pattern	pre-DNS syntax check
dns_ipv4_lookup	only v4 lookup for these domains
dns_retrans	parameter for resolver
dns_retry	parameter for resolver
hold_domains	hold delivery for these domains
local_interfaces	for routing checks
queue_domains	no immediate delivery for these
queue_only	no immediate delivery at all
queue_only_file	no immediate deliveryif file exists
queue_only_load	no immediate delivery if load is high
queue_run_in_order	order of arrival
queue_run_max	of simultaneous queue runners
queue_smtp_domains	no immediate SMTP delivery for these
remote_max_parallel	parallel SMTP delivery per message
remote_sort_domains	order of remote deliveries
retry_data_expire	timeout for retry data
retry_interval_max	safety net for retry rules

13.21 Bounce and warning messages

bounce_message_file	content of bounce
bounce_message_text	content of bounce
bounce_return_message	include original message in bounce
bounce_sender_authentication	send authenticated sender with bounce
errors_copy	copy bounce messages
errors_reply_to	*Reply-to:* in bounces
delay_warning	time schedule
delay_warning_condition	condition for warning messages
ignore_bounce_errors_after	discard undeliverable bounces
return_size_limit	limit on returned message
warn_message_file	content of warning message

13.22 Alphabetical list of main options

Those options that undergo string expansion before use are marked with †.

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.

acl_not_smtp Type: *string*† Default: *unset*

This option defines the ACL that is run when a non-SMTP message is on the point of being accepted. See chapter 37 for further details.

acl_smtp_connect Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP connection is received. See chapter 37 for further details.

acl_smtp_auth Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP AUTH command is received. See chapter 37 for further details.

acl_smtp_data Type: *string*† Default: *unset*

This option defines the ACL that is run after an SMTP DATA command has been processed and the message itself has been received, but before the final acknowledgement is sent. See chapter 37 for further details.

acl_smtp_etrn Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP ETRN command is received. See chapter 37 for further details.

acl_smtp_expn Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP EXPN command is received. See chapter 37 for further details.

acl_smtp_helo Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP EHLO or HELO command is received. See chapter 37 for further details.

acl_smtp_mail Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP MAIL command is received. See chapter 37 for further details.

acl_smtp_rcpt Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP RCPT command is received. See chapter 37 for further details.

acl_smtp_starttls Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP STARTTLS command is received. See chapter 37 for further details.

acl_smtp_vrfy Type: *string*† Default: *unset*

This option defines the ACL that is run when an SMTP VRFY command is received. See chapter 37 for further details.

admin_groups Type: *string list* Default: *unset*

If the current group or any of the supplementary groups of the caller is in this colon-separated list, the caller has admin privileges. If all your system programmers are in a specific group, for example, you can give them all Exim admin privileges by putting that group in **admin_groups**. However, this does not permit them to read Exim's spool files (whose group owner is the Exim gid). To permit this, you have to add individuals to the Exim group.

allow_domain_literals Type: *boolean* Default: *false*

If this option is set, the RFC 2822 domain literal format is permitted in email addresses. The option is not set by default, because the domain literal format is not normally required these days, and few people know about it. It has, however, been exploited by mail abusers.

Unfortunately, it seems that some DNS black list maintainers are using this format to report black listing to postmasters. If you want to accept messages addressed to your hosts by IP address, you need to set **allow_domain_literals** true, and also to add `@[]` to the list of local domains (defined in the named domain list **local_domains** in the default configuration). This 'magic string' matches the domain literal form of all the local host's IP addresses.

allow_mx_to_ip Type: *boolean* Default: *false*

It appears that more and more DNS zone administrators are breaking the rules and putting domain names that look like IP addresses on the right hand side of MX records. Exim follows the rules and rejects this, giving an error message that explains the mis-configuration. However, some other MTAs support this practice, so to avoid 'Why can't Exim do this?' complaints, **allow_mx_to_ip** exists, in order to enable this heinous activity. It is not recommended, except when you have no other choice.

allow_utf8_domains Type: *boolean* Default: *false*

Lots of discussion is going on about internationalized domain names. One camp is strongly in favour of just using UTF-8 characters, and it seems that at least two other MTAs permit this. This option allows Exim users to experiment if they wish.

If it is set true, Exim's domain parsing function allows valid UTF-8 multicharacters to appear in domain name components, in addition to letters, digits, and hyphens. However, just setting this option is not enough; if you want to look up these domain names in the DNS, you must also adjust the value of **dns_check_names_pattern** to match the extended form. A suitable setting is:

```
dns_check_names_pattern = (?i)^(?>(?(1)\.|())[a-z0-9\xc0-\xff]\
  (?>[-a-z0-9\x80-\xff]*[a-z0-9\x80-\xbf])?)+$
```

Alternatively, you can just disable this feature by setting

```
dns_check_names_pattern =
```

That is, set the option to an empty string so that no check is done.

auth_advertise_hosts Type: *host list*† Default: *

If any server authentication mechanisms are configured, Exim advertises them in response to an EHLO command only if the calling host matches this list. Otherwise, Exim does not advertise AUTH. Exim does not accept AUTH commands from clients to which it has not advertised the availability of AUTH. The advertising of individual authentication mechanisms can be controlled by the use of the **server_advertise_condition** generic authenticator option on the individual authenticators. See chapter 32 for further details.

Certain mail clients (for example, Netscape) require the user to provide a name and password for authentication if AUTH is advertised, even though it may not be needed (the host may accept messages from hosts on its local LAN without authentication, for example). The **auth_advertise_hosts** option can be used to make these clients more friendly by excluding them from the set of hosts to which Exim advertises AUTH.

If you want to advertise the availability of AUTH only when the connection is encrypted using TLS, you can make use of the fact that the value of this option is expanded, with a setting like this:

```
auth_advertise_hosts = ${if eq{$tls_cipher}{}{}{*}}
```

If **$tls_cipher** is empty, the session is not encrypted, and the result of the expansion is empty, thus matching no hosts. Otherwise, the result of the expansion is *, which matches all hosts.

auto_thaw Type: *time* Default: *0s*

If this option is set to a time greater than zero, a queue runner will try a new delivery attempt on any frozen message if this much time has passed since it was frozen. This may result in the message being re-frozen if nothing has changed since the last attempt. It is a way of saying 'keep on trying, even though there are big problems'. See also **timeout_frozen_after** and **ignore_bounce_errors_after**.

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.

bounce_message_file Type: *string* Default: *unset*

This option defines a template file containing paragraphs of text to be used for constructing bounce messages. Details of the file's contents are given in chapter 40. See also **warn_message_file**.

bounce_message_text Type: *string* Default: *unset*

When this option is set, its contents are included in the default bounce message immediately after 'This message was created automatically by mail delivery software.' It is not used if **bounce_message_file** is set.

bounce_return_message Type: *boolean* Default: *true*

If this option is set false, the original message is not included in bounce messages generated by Exim. See also **return_size_limit**.

bounce_sender_authentication Type: *string* Default: *unset*

This option provides an authenticated sender address that is sent with any bounce messages generated by Exim that are sent over an authenticated SMTP connection. A typical setting might be:

```
bounce_sender_authentication = mailer-daemon@my.domain.example
```

which would cause bounce messages to be sent using the SMTP command:

```
MAIL FROM:<> AUTH=mailer-daemon@my.domain.example
```

The value of **bounce_sender_authentication** must always be a complete email address.

callout_domain_negative_expire Type: *time* Default: *3h*

This option specifies the expiry time for negative callout cache data for a domain. See section 37.14 for details of callout verification, and section 37.16 for details of the caching.

callout_domain_positive_expire Type: *time* Default: *7d*

This option specifies the expiry time for positive callout cache data for a domain. See section 37.14 for details of callout verification, and section 37.16 for details of the caching.

callout_negative_expire Type: *time* Default: *2h*

This option specifies the expiry time for negative callout cache data for an address. See section 37.14 for details of callout verification, and section 37.16 for details of the caching.

callout_positive_expire Type: *time* Default: *24h*

This option specifies the expiry time for positive callout cache data for an address. See section 37.14 for details of callout verification, and section 37.16 for details of the caching.

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 disk 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**. It is used for holding messages in transit.

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** and **spool_directory** refer to different partitions.

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.

daemon_smtp_port Type: *string* Default: *unset*

This option specifies the default SMTP port on which the Exim daemon listens. It can either be given as a number, or as a service name. It can be overridden by giving an explicit port number on an IP address in the **local_interfaces** option, or by using **-oX** on the command line. If this option is not set, the service name ‘smtp’ is used.

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 the third one after 24 hours. After that, messages are sent every 16 hours, because that is the interval between the last two times on the list. If you set just one time, it specifies the repeat interval. For example, with:

```
delay_warning = 6h
```

messages are repeated every six hours. To stop warnings after a given time, set a very large time at the the end of the list. For example:

```
delay_warning = 2h:12h:99d
```

delay_warning_condition Type: *string†* Default: *see below*

The string is expanded at the time a warning message might be sent. If all the deferred addresses have the same domain, it is set in **$domain** during the expansion. Otherwise **$domain** is empty. 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. The default is

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

which suppresses the sending of warnings about messages that have ‘bulk’, ‘list’ or ‘junk’ in a *Precedence:* header.

deliver_drop_privilege Type: *boolean* Default: *false*

If this option is set true, Exim drops its root privilege at the start of a delivery process, and runs as the Exim user throughout. This severely restricts the kinds of local delivery that are possible, but is viable in certain types of configuration. There is a discussion about the use of root privilege in chapter 47.

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

When this option is set, a queue run is abandoned if the system load average becomes greater than the value of the option. The option has no effect on ancient operating systems on which Exim cannot determine the load average. See also **queue_only_load** and **smtp_load_reserve**.

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 at the time the message is received, 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 errors ‘non-authoritative host not found’ and ‘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 name server and may persist for a long time. If a domain which exhibits this problem matches anything in **dns_again_means_nonexist**, it is treated as if it did not exist. This option should be used with care.

dns_check_names_pattern Type: *string* Default: *see below*

When this option is set to a non-empty string, it causes Exim to check domain names for illegal characters before handing them to the DNS resolver, because some resolvers give temporary errors for malformed names. If a domain name contains any illegal characters, a ‘not found’ result is forced, and the resolver is not called. The check is done by matching the domain name against a regular expression, which is the value of this option. The default pattern 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. If you set **allow_utf8_domains**, you must modify this pattern, or set the option to an empty string.

dns_ipv4_lookup Type: *domain list†* Default: *unset*

When Exim is compiled with IPv6 support, it looks for IPv6 address records (AAAA and, if configured, A6) as well as IPv4 address records when trying to find IP addresses for hosts, unless the host's domain matches this list.

This is a fudge to help with name servers that give big delays or otherwise do not work for the new IPv6 record types. If Exim is handed an IPv6 address record as a result of an MX lookup, it always recognizes it, and may as a result make an outgoing IPv6 connection. All this option does is to make Exim look only for IPv4-style A records when it needs to find an IP address for a host name. In due course, when the world's name servers have all been upgraded, there should be no need for this option.

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.

drop_cr Type: *boolean* Default: *false*

Setting **drop_cr** true affects non-SMTP messages that are submitted locally. It causes every carriage return character that immediately precedes a linefeed to be discarded. Other carriage returns are treated as data. This action can be requested for individual messages by means of the **-dropcr** command line option.

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 from the messages's envelope that caused the delivery to happen. Such headers should not be present in incoming messages, and this option causes them to be removed at the time the message is received, to avoid any problems that might occur when a delivered message is subsequently sent on to some other recipient.

errors_copy Type: *string list†* Default: *unset*

Setting this option causes Exim to send bcc copies of bounce messages that it generates to other addresses. **Note**: this does not apply to bounce messages coming from elsewhere. The value of the option is a colon-separated list of items. Each item consists of a pattern, terminated by white space, followed by a comma-separated list of email addresses. If a pattern contains spaces, it must be enclosed in double quotes.

Each pattern is processed in the same way as a single item in an address list (see section 10.13). When a pattern matches the recipient of the bounce message, 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.example :\
              rqps@mydomain   hostmaster@mydomain.example,\
                              postmaster@mydomain.example
```

The address list is expanded before use. The expansion variables **$local_part** and **$domain** are set from the original recipient of the error message, and if there was any wildcard matching in the pattern, the expansion variables **$0**, **$1**, etc. are set in the normal way.

errors_reply_to Type: *string* Default: *unset*

Exim's bounce and delivery warning messages contain the header line

```
From: Mail Delivery System <Mailer-Daemon@<qsdomain>>
```

where <*qsdomain*> is the value of **qualify_domain_sender**. Experience shows that a large number of people reply to bounce messages. If the **errors_reply_to** option is set, a *Reply-To:* header is added to bounce and warning messages. For example:

```
errors_reply_to = postmaster@my.domain.example
```

The value of the option is not expanded. It must specify a valid RFC 2822 address.

exim_group Type: *string* Default: *compile-time configured*

This option changes the gid under which Exim runs when it gives up root privilege. The default value is compiled into the binary. The value of this option 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 47 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, exceptionally, Exim is run from some other place. **Warning**: Do not use a macro to define the value of this option, because you will break those Exim utilities that scan the configuration file to find where the binary is. (They then use the **-bP** option to extract option settings such as the value of **spool_directory**.)

exim_user Type: *string* Default: *compile-time configured*

This option changes the uid under which Exim runs when it gives up root privilege. The default value is compiled into the binary. Ownership of the run time configuration file and the use of the **-C** and **-D** command line options is checked against the values in the binary, not what is set here.

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. See chapter 47 for a discussion of security issues.

extract_addresses_remove_arguments Type: *boolean* Default: *true*

According to some Sendmail documentation (Sun, IRIX, HP-UX), if any addresses are present on the command line when the **-t** option is used to build an envelope from a message's *To:*, *Cc:* and *Bcc:* headers, the command line addresses are removed from the recipients list. This is also how Smail behaves. However, other Sendmail documentation (the O'Reilly book) states that command line addresses are added to those obtained from the header lines. When **extract_addresses_remove_arguments** is true (the default), Exim subtracts argument headers. If it is set false, Exim 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 retries.

freeze_tell Type: *string list, comma separated* Default: *unset*

On encountering certain errors, or when configured to do so in a system filter, Exim freezes a message. This means that no further delivery attempts take place until an administrator (or the **auto_thaw** feature) thaws the message. If **freeze_tell** is set, Exim generates a warning message whenever it freezes something, unless the message it is freezing is a bounce message. (Without this exception there is the possibility of looping.) The warning message is sent to the addresses supplied

as the comma-separated value of this option. If several of the message's addresses cause freezing, only a single message is sent. The reason(s) for freezing can 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.

header_maxsize Type: *integer* Default: *see below*

This option controls the overall maximum size of a message's header section. The default is the value of HEADER_MAXSIZE in **Local/Makefile**; the default for that is 1M. Messages with larger header sections are rejected.

header_line_maxsize Type: *integer* Default: *0*

This option limits the length of any individual header line in a message, after all the continuations have been joined together. Messages with individual header lines that are longer than the limit are rejected. The default value of zero means 'no limit'.

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. Note that this is a syntax check only. See **helo_verify_hosts** if you want to do semantic checking. See also **helo_allow_chars** for a way of extending the permitted character set.

helo_allow_chars Type: *string* Default: *unset*

This option can be set to a string of rogue characters that are permitted in all EHLO and HELO names in addition to the standard letters, digits, hyphens, and dots. If you really must allow underscores, you can set

```
helo_allow_chars = _
```

Note that the value is one string, not a list.

helo_lookup_domains Type: *domain list†* Default: `@:@[]`

If the domain given by a client in a HELO or EHLO command matches this list, a reverse lookup is done in order to establish the host's true name. The default forces a lookup if the client host gives the server's name or any of its IP addresses (in brackets), something that broken clients have been seen to do.

helo_try_verify_hosts 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. By default, Exim just checks the syntax of these commands (see **helo_accept_junk_hosts** and **helo_allow_chars** above). However, some sites like to be stricter. If the calling host matches **helo_try_verify_hosts**, Exim checks that the host name given in the HELO or EHLO command either:

- is an IP literal matching the calling address of the host (the RFCs specifically allow this), or
- matches the host name that Exim obtains by doing a reverse lookup of the calling host address, or
- when looked up using *gethostbyname()* (or *getipnodebyname()* when available) yields the calling host address.

However, the EHLO or HELO command is not rejected if any of the checks fail. Processing continues, but the result of the check is remembered, and can be detected later in an ACL by the `verify = helo` condition. If you want verification failure to cause rejection of EHLO or HELO, use **helo_verify_hosts** instead.

helo_verify_hosts Type: *host list†* Default: *unset*

For hosts that match this option, Exim checks the host name given in the HELO or EHLO in the same way as for **helo_try_verify_hosts**. If the check fails, the HELO or EHLO command is rejected with a 550 error, and entries are written to the main and reject logs. If a MAIL command is received before EHLO or HELO, it is rejected with a 503 error.

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**, **-qf**, **-Rf** or **-Sf** options, and also while testing or verifying addresses using **-bt** or **-bv**. 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. If you just want to delay the processing of some domains until a queue run occurs, you should use **queue_domains** or **queue_smtp_domains**, not **hold_domains**.

A setting of **hold_domains** does not override Exim's code for removing 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_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 the host matches **helo_try_verify_hosts** or **helo_verify_hosts**, or the host matches this option (which normally contains IP addresses rather than host names). The default configuration file contains

```
host_lookup = *
```

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.

After a successful reverse lookup, Exim does a forward lookup on the name it has obtained, to verify that it yields the IP address that it started with. If this check fails, Exim behaves as if the name lookup failed.

After any kind of failure, the host name (in **$sender_host_name**) remains unset, and **$host_lookup_failed** is set to the string '1'. See also **helo_lookup_domains** and `verify = reverse_host_lookup` in ACLs.

host_reject_connection Type: *host list†* Default: *unset*

If this option is set, incoming SMTP calls from the hosts listed are rejected as soon as the connection is made. The ACL specified by **acl_smtp_connect** can also reject incoming connections immediately. The ability to give an immediate rejection is provided for use in unusual cases. Many hosts will just try again, sometimes without much delay. Normally, it is better to use an ACL to reject incoming messages at a later stage, such as after RCPT commands. See chapter 37.

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. This option also applies when Exim is matching the special items `@mx_any`, `@mx_primary`, and `@mx_secondary` in a domain list (see section 10.7), and when checking the **hosts** option in the **smtp** transport for the local host (see the **allow_localhost** option in that transport).

ignore_bounce_errors_after Type: *time* Default: *10w*

This option affects the processing of bounce messages that cannot be delivered. After an initial failure, such messages are frozen, because there is no sender to whom they can be returned. When a frozen bounce message has been on the queue for more than the given time, it is unfrozen at the next queue run, and a further delivery is attempted. If delivery fails again, the bounce message is discarded. This makes it possible to keep failed bounce messages around for a shorter time than the normal maximum retry time for frozen messages. For example,

```
ignore_bounce_errors_after = 12h
```

retries failed bounce message deliveries after 12 hours, discarding any further failures. If the value of this option is set to a zero time period, bounce failures are discarded immediately. Setting a very long time (as in the default value) has the effect of disabling this option. For ways of automatically dealing with other kinds of frozen message, see **auto_thaw** and **timeout_frozen_after**.

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, **ignore_fromline_local** must be set to achieve this effect.

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.

ldap_default_servers Type: *string list* Default: *unset*

This option provides a list of LDAP servers which are tried in turn when an LDAP query does not contain a server. See section 9.11 for details of LDAP queries. This option is available only when Exim has been built with LDAP support.

ldap_version Type: *integer* Default: *unset*

This option can be used to force Exim to set a specific protocol version for LDAP. If it option is unset, it is shown by the **-bP** command line option as -1. When this is the case, the default is 3 if LDAP_VERSION3 is defined in the LDAP headers; otherwise it is 2. This option is available only when Exim has been built with LDAP support.

local_from_check Type: *boolean* Default: *true*

When a message is submitted locally (that is, not over a TCP/IP connection) by an untrusted user, Exim removes any existing *Sender:* header line, and checks that the *From:* header line matches the login of the calling user. You can use **local_from_prefix** and **local_from_suffix** to permit affixes on the local part. If the *From:* header line does not match, Exim adds a *Sender:* header with an address constructed from the calling user's login and the default qualify domain.

If **local_from_check** is set false, the *From:* header check is disabled, and no *Sender:* header is ever added. If, in addition, you want to retain *Sender:* header lines supplied by untrusted users, you must also set **local_sender_retain** to be true.

These options affect only the header lines in the message. The envelope sender is still forced to be the login id at the qualify domain unless **untrusted_set_sender** permits the user to supply an envelope sender. Section 43.12 has more details about *Sender:* processing.

local_from_prefix Type: *string* Default: *unset*

When Exim checks the *From:* header line of locally submitted messages for matching the login id (see **local_from_check** above), it can be configured to ignore certain prefixes and suffixes in the local part of the address. This is done by setting **local_from_prefix** and/or **local_from_suffix** to appropriate lists, in the same form as the **local_part_prefix** and **local_part_suffix** router options (see chapter 14). For example, if

```
local_from_prefix = *-
```

is set, a *From:* line containing

```
From: anything-user@your.domain.example
```

will not cause a *Sender:* header to be added if *user@your.domain.example* matches the actual sender address that is constructed from the login name and qualify domain.

local_from_suffix Type: *string* Default: *unset*

See **local_from_prefix** above.

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 for IPv6 addresses. It is usually easier to change the list separator character instead of doubling all the colons in IPv6 addresses. For example:

```
local_interfaces = <; 127.0.0.1 ; \
                      192.168.23.65 ; \
                      ::1 ; \
                      3ffe:ffff:836f::fe86:a061
```

IPv6 addresses have 'scopes', and a host with multiple interfaces can, in principle, have the same link-local addresses on different interfaces. Thus, they need to be distinguished, and a convention of using a percent sign followed by something (often the interface name) has been adopted in some cases, leading to addresses like this:

```
3ffe:2101:12:1:a00:20ff:fe86:a061%eth0
```

To accommodate this usage, a percent sign followed by an arbitrary string is allowed at the end of an IPv6 address. By default, Exim calls *getaddrinfo()* to convert a textual IPv6 address for actual use. This function recognizes the percent convention in operating systems that support it, and it processes the address appropriately.

Unfortunately, some older libraries have problems with *getaddrinfo()*. If

```
IPV6_USE_INET_PTON=yes
```

is set in **Local/Makefile** (or an OS-dependent Makefile) when Exim is built, Exim uses *inet_pton()* to convert a textual IPv6 address for actual use, instead of *getaddrinfo()*. (Before version 4.14, it

always used this function.) Of course, this means that the additional functionality of *getaddrinfo()* – recognizing scoped addresses – is lost.

A port number can be specified along with each IP address. Two different formats are recognized:

- The port is added onto the address with a dot separator, for example,

```
local_interfaces = <; 192.168.23.65.1234 ; \
                      3ffe:ffff:836f::fe86:a061.1234
```

- The IP address is enclosed in square brackets, and the port is added with a colon separator, for example,

```
local_interfaces = <; [192.168.23.65]:1234 ; \
                      [3ffe:ffff:836f::fe86:a061]:1234
```

The list of IP addresses in **local_interfaces** is used for two different purposes:

- When a daemon is started to listen for incoming SMTP calls, it listens only on the interfaces and ports identified here. For interfaces listed without a port, the value of **daemon_smtp_port** is used, unless overridden by the **-oX** option. Exim can be made to listen on more than one port by listing the same interface with different port numbers, for example:

```
local_interfaces = 192.168.34.67.25 : 192.168.34.67.26
```

  The address 0.0.0.0 means 'any interface' in most IPv4 stacks, and ::0 is the IPv6 equivalent. So, to specify listening on multiple ports on any interface, you can use settings such as:

```
local_interfaces = <; 0.0.0.0.25 ; 0.0.0.0.26 ; \
                      ::0.25     ; ::0.26
```

  When a message is received over TCP/IP, the interface and port that were used are set in **$interface_address** and **$interface_port**. When the daemon is started, an error occurs if it is unable to bind a listening socket to any listed interface.

- Only the IP addresses listed here are taken as the local host's IP addresses when routing mail and checking for mail loops. In this case, the port values are not used.

If **local_interfaces** is unset, the daemon issues a generic *listen()* that accepts incoming calls to any interface. The same port is used in all cases. In addition, Exim gets a complete list of available interfaces from the operating system, 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**.

local_scan_timeout Type: *time* Default: *5m*

This timeout applies to the *local_scan()* function (see chapter 38). Zero means 'no timeout'. If the timeout is exceeded, the incoming message is rejected with a temporary error if it is an SMTP message. For a non-SMTP message, the message is dropped and Exim ends with a non-zero code. The incident is logged on the main and reject logs.

local_sender_retain Type: *boolean* Default: *false*

When a message is submitted locally (that is, not over a TCP/IP connection) by an untrusted user, Exim removes any existing *Sender:* header line. If you do not want this to happen, you must set **local_sender_retain**, and you must also set **local_from_check** to be false (Exim will complain if you do not). Section 43.12 has more details about *Sender:* processing.

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, 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 0–16 (or 0–10 on operating systems with case-insensitive file systems). This is available in subsequent string expansions via the variable **$localhost_number**. When **localhost_number is set**, the final two characters of the message id, instead of just being a fractional part of the time, are computed from the time and the local host number as described in section 3.3.

log_file_path Type: *string list†* Default: *set at compile time*

This option sets the path which is used to determine the names of Exim's log files, or indicates that logging is to be to syslog, or both. It is expanded when Exim is entered, so it can, for example, contain a reference to the host name. If no specific path is set for the log files at compile or run time, they are written in a sub-directory called **log** in Exim's spool directory. Chapter 44 contains further details about Exim's logging, and section 44.1 describes how the contents of **log_file_path** are used. If this 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.

log_selector Type: *string* Default: *unset*

This option can be used to reduce or increase the number of things that Exim writes to its log files. Its argument is made up of names preceded by plus or minus characters. For example:

```
log_selector = +arguments -retry_defer
```

A list of possible names and what they control is given in the chapter on logging, in section 44.15.

log_timezone Type: *boolean* Default: *false*

By default, the timestamps on log lines are in local time without the timezone. This means that if your timezone changes twice a year, the timestamps in log lines are ambiguous for an hour when the clocks go back. One way of avoiding this problem is to set the timezone to UTC. An alternative is to set **log_timezone** true. This turns on the addition of the timezone offset to timestamps in log lines. Turning on this option can add quite a lot to the size of log files because each line is extended by 6 characters. Note that the **$tod_log** variable contains the log timestamp without the zone, but there is another variable called **$tod_zone** that contains just the timezone offset.

lookup_open_max Type: *integer* Default: *25*

This option limits the number of simultaneously open files for single-key lookups that use regular files (that is, **lsearch**, **dbm**, and **cdb**). Exim normally keeps these files open during routing, because often the same file is required several times. 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_username_length Type: *integer* Default: *0*

Some operating systems are broken in that they truncate long arguments 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 **$message_body** and **$message_body_end** expansion variables.

message_id_header_domain Type: *string†* Default: *unset*

If this option is set, the string is expanded and used as the right hand side (domain) of the *Message-ID:* header that Exim creates if an incoming message does not have one. Otherwise, the primary host name is used. Only letters, digits, dot and hyphen are accepted; any other characters are replaced by hyphens. If the expansion is forced to fail, or if the result is an empty string, the option is ignored.

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 2822 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 is not forced to fail, and does not yield an empty string, the result 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 variables such as **$tod_log** can be used, because the spaces and colons will become hyphens.

message_logs Type: *boolean* Default: *true*

If this option is turned off, per-message log files are not created in the **msglog** spool sub-directory. This reduces the amount of disk I/O required by Exim, by reducing the number of files involved in handling a message from a minimum of four (header spool file, body spool file, delivery journal, and per-message log) to three. The other major I/O activity is Exim's main log, which is not affected by this option.

message_size_limit Type: *string†* Default: *50M*

This option limits the maximum size of message that Exim will process. The value is expanded for each incoming connection so, for example, it can be made to depend on the IP address of the remote host for messages arriving via TCP/IP. **Note**: This limit cannot be made to depend on a message's sender or any other properties of an individual message, because it has to be advertised in the server's response to EHLO. String expansion failure causes a temporary error. A value of zero means no limit, but its use is not recommended. See also **return_size_limit**.

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. 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.

move_frozen_messages Type: *boolean* Default: *false*

This option, which is available only if Exim has been built with the setting

```
SUPPORT_MOVE_FROZEN_MESSAGES=yes
```

in **Local/Makefile**, causes frozen messages and their message logs to be moved from the **input** and **msglog** directories on the spool to **Finput** and **Fmsglog**, respectively. There is currently no support in Exim or the standard utilities for handling such moved messages, and they do not show up in lists generated by **-bp** or by the Exim monitor.

mysql_servers Type: *string list* Default: *unset*

This option provides a list of MySQL servers and associated connection data, to be used in conjunction with **mysql** lookups (see section 9.17). The option is available only if Exim has been built with MySQL support.

never_users Type: *string list* Default: *unset*

Local mail deliveries are normally run in processes that are setuid to the recipient, and remote deliveries are normally run under Exim's own uid and gid. It is usually desirable to prevent any deliveries from running as root, as a safety precaution. If a message is to be delivered as one of the users on the **never_users** list, an error occurs, and delivery is deferred. A common example is

```
never_users = root:daemon:bin
```

This option overrides the **pipe_as_creator** option of the **pipe** transport driver.

oracle_servers Type: *string list* Default: *unset*

This option provides a list of Oracle servers and associated connection data, to be used in conjunction with **oracle** lookups (see section 9.17). The option is available only if Exim has been built with Oracle support.

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 new email address, with the percent replaced by @. This is sometimes called 'source routing', though that term is also applied to RFC 2822 addresses that begin with an @ character. If this option is set, Exim implements the percent facility for those domains listed, but no others. This happens before an incoming SMTP address is tested against an ACL.

Warning: The 'percent hack' has often been abused by people who are trying to get round relaying restrictions. For this reason, it is best avoided if at all possible. Unfortunately, a number of less security-conscious MTAs implement it unconditionally. If you are running Exim on a gateway host, and routing mail through to internal MTAs without processing the local parts, it is a good idea to reject recipient addresses with percent characters in their local parts. Exim's default configuration does this.

perl_at_start Type: *boolean* Default: *false*

This option is available only when Exim is built with an embedded Perl interpreter. See chapter 12 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 12 for details of its use.

pgsql_servers Type: *string list* Default: *unset*

This option provides a list of PostgreSQL servers and associated connection data, to be used in conjunction with **pgsql** lookups (see section 9.17). The option is available only if Exim has been built with PostgreSQL support.

pid_file_path Type: *string†* Default: *set at compile time*

This option sets 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:

```
pid_file_path = /var/log/$primary_hostname/exim.pid
```

If no path is set, the pid is written to the file **exim-daemon.pid** in Exim's spool directory. The value set by the option can be overridden by the **-oP** command line option. A pid file is not written if a 'non-standard' daemon is run by means of the **-oX** option, unless a path is explicitly supplied by **-oP**.

pipelining_advertise_hosts Type: *host list†* Default: *

This option can be used to suppress the advertisement of the SMTP PIPELINING extension to specific hosts. When PIPELINING is not advertised and **smtp_enforce_sync** is true, an Exim server enforces strict synchronization for each SMTP command and response.

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 EHLO 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()* (or *getipnodebyname()* when available) 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**.

qualify_domain Type: *string* Default: *see below*

This option 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 **sender_unqualified_hosts** or **recipient_unqualified_hosts** (as appropriate), in which case incoming addresses are qualified with **qualify_domain** or **qualify_recipient** as necessary. Internally, Exim always works with fully qualified addresses. 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 recipient addresses, unless the sending host matches **recipient_unqualified_hosts**, in which case incoming recipient addresses are qualified with **qualify_recipient**. If **qualify_recipient** is not set, it defaults to the **qualify_domain** value.

queue_domains Type: *domain list†* Default: *unset*

This option lists domains for which immediate delivery is not required. A delivery process is started whenever a message is received, but only those domains that do not match are processed. All other deliveries wait until the next queue run. See also **hold_domains** and **queue_smtp_domains**.

queue_list_requires_admin Type: *boolean* Default: *true*

The **-bp** command-line option, which lists the messages that are on the queue, requires the caller to be an admin user unless **queue_list_requires_admin** is set false. See also **prod_requires_admin**.

queue_only Type: *boolean* Default: *false*

If **queue_only** is set, a delivery process is not automatically started whenever a message is received. Instead, the message waits on the queue for the next queue run. The **-odq** command line has the same effect. Even if **queue_only** is false, incoming messages may not get delivered immediately when certain conditions occur. See **queue_only_file**, **queue_only_load** and **smtp_accept_queue**.

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 ‘smtp’. When Exim is receiving a message, it tests for the existence of each listed path using a call to *stat()*. For each path that exists, the corresponding queuing option is set. For paths with no prefix, **queue_only** is set; for paths prefixed by ‘smtp’, **queue_smtp_domains** is set to match all domains. So, for example,

```
queue_only_file = smtp/some/file
```

causes Exim to behave as if **queue_smtp_domains** 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, incoming messages from all sources 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 runner processes. This option has no effect on ancient operating systems on which Exim cannot determine the load average. See also **deliver_queue_load_max** and **smtp_load_reserve**.

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. For this to happen, a complete list of the entire queue must be set up before the deliveries start. When the queue is all in a single directory (the default), this happens anyway, but if **split_spool_directory** is set it does not – for delivery in random order, the sub-directories are processed one at a time (in random order), to avoid setting up one huge list. Thus, setting **queue_run_in_order** with **split_spool_directory** may degrade performance when the queue is large. In most situations, **queue_run_in_order** should not be set.

queue_run_max Type: *integer* Default: *5*

This controls the maximum number of queue runner processes that an Exim daemon can 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 another one. 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, 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 message 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. 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 **hold_domains** and **queue_domains**.

receive_timeout Type: *time* Default: *0s*

This option 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 line option. The timeout for incoming SMTP messages is controlled by **smtp_receive_timeout**.

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}} \
    ${if def:tls_cipher {($tls_cipher)\n\t}}\
    (Exim $version_number)\n\t\
    id $message_id\
    ${if def:received_for {\n\tfor $received_for}}
```

Note the use of quotes, to allow the sequences `\n` and `\t` to be used for newlines and tabs, respectively. The reference to the TLS cipher is omitted when Exim is built without TLS support. 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.example ([192.168.12.25] ident=root)
        by marley.carol.example with esmtp (Exim 4.00)
        id 16IOWa-00019l-00
        for chas@dickens.example; Tue, 25 Dec 2001 14:43:44 +0000
Received: by scrooge.carol.example with local (Exim 4.00)
        id 16IOWW-000083-00; Tue, 25 Dec 2001 14:43:41 +0000
```

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.

recipient_unqualified_hosts Type: *host list†* Default: *unset*

This option lists those hosts from which Exim is prepared to accept unqualified recipient addresses in message envelopes. The addresses are made fully qualified by the addition of the **qualify_recipient** value. This option also affects message header lines. Exim does not reject unqualified recipient addresses in headers, but it qualifies them only if the message came from a host that matches **recipient_unqualified_hosts**.

recipients_max Type: *integer* Default: *0*

If this option is set greater than zero, it specifies the maximum number of original recipients for any message. Additional recipients that are generated by aliasing or forwarding do not count. 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, 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.

remote_max_parallel Type: *integer* Default: *2*

This option controls parallel delivery of one message to a number of remote hosts. If the value is less than 2, parallel delivery is disabled, and Exim does all the remote deliveries for a message one by one. Otherwise, if a single message has to be delivered to more than one remote host, or if several copies have to be sent to the same remote host, up to **remote_max_parallel** deliveries are done simultaneously. If more than **remote_max_parallel** deliveries are required, 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_domains** 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.

This option controls only the maximum number of parallel deliveries for one message in one Exim delivery process. Because 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, you need to set the **queue_only** option. This ensures that all incoming messages are added to the queue without starting a delivery process. 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. Because 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, 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_domains 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_domains = *.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_data_expire Type: *time* Default: *7d*

This option sets a ‘use before’ time on retry information in Exim’s hints database. Any older retry data is ignored. This means that, for example, once a host has not been tried for 7 days, Exim behaves as if it has no knowledge of past failures.

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 2821, section 4.4, states that an SMTP server must insert a *Return-path:* header line into a message when it makes a ‘final delivery’. The *Return-path:* header preserves the sender address as received in the MAIL command. This description implies that this header should not be present in an incoming message. If **return_path_remove** is true, any existing *Return-path:* headers are removed from messages at the time they are received. 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 as part of bounce messages. The limit should be less than the value of the global **message_size_limit** and of any **message_size_limit** settings on transports, to allow for the bounce text that Exim generates. If this option is set to zero there is no limit.

When the body of any message that is to be included in a bounce message 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 (typically 8K in size). The idea is just to save bandwidth on those undeliverable 15-megabyte messages. See also **bounce_return_message**.

rfc1413_hosts Type: *host list†* Default: *

RFC 1413 identification calls are made to any client host which matches an item in the list.

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.

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**. This option also affects message header lines. Exim does not reject unqualified addresses in headers that contain sender addresses, but it qualifies them only if the message came from a host that matches **sender_unqualified_hosts**.

smtp_accept_keepalive Type: *boolean* Default: *true*

This option controls the setting of the SO_KEEPALIVE option on incoming TCP/IP socket connections. When set, it causes the kernel to probe idle connections periodically, by sending packets with ‘old’ sequence numbers. The other end of the connection should send an acknowledgement if the connection is still okay or a reset if the connection has been aborted. 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. The keepalive mechanism takes several hours to detect unreachable hosts.

smtp_accept_max Type: *integer* Default: *20*

This option 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 either **smtp_accept_max_per_host** or **smtp_accept_queue** is set. See also **smtp_accept_reserve**.

smtp_accept_max_nonmail Type: *integer* Default: *10*

Exim counts the number of ‘non-mail’ commands in an SMTP session, and drops the connection if there are too many. This option defines ‘too many’. The check catches some denial-of-service attacks. repeated failing AUTHs, or a mad client looping sending EHLO, for example. The check is applied only if the client host matches **smtp_accept_max_nonmail_hosts**.

When a new message is expected, one occurrence of RSET is not counted. This allows a client to send one RSET between messages (this is not necessary, but some clients do it). Exim also allows one uncounted occurence of HELO or EHLO, and one occurrence of STARTTLS between messages. After starting up a TLS session, another EHLO is expected, and so it too is not counted. The first occurrence of AUTH in a connection, or immediately following STARTTLS is not counted. Otherwise, all commands other than MAIL, RCPT, DATA, and QUIT are counted.

smtp_accept_max_nonmail_hosts Type: *host list*† Default: *

You can control which hosts are subject to the **smtp_max_nonmail** check by setting this option. The default value makes it apply to all hosts. By changing the value, you can exclude any badly-behaved hosts that you have to live with.

smtp_accept_max_per_connection Type: *integer* Default: *1000*

The value of this option limits the number of MAIL commands that Exim is prepared to accept over a single SMTP connection, whether or not each command results in the transfer of a message. After the limit is reached, a 421 response is given to subsequent MAIL commands. This limit is a safety precaution against a client that goes mad (incidents of this type have been seen).

smtp_accept_max_per_host Type: *string*† Default: *unset*

This option restricts the number of simultaneous IP connections from a single host (strictly, from a single IP address) to the Exim daemon. The option is expanded, to enable different limits to be applied to different hosts by reference to **$sender_host_address**. Once the limit is reached, additional connection attempts from the same host are rejected with error code 421. The default value of zero imposes no limit. If this option is set, it is required that **smtp_accept_max** be non-zero.

Warning: When setting this option you should not use any expansion constructions that take an appreciable amount of time. The expansion and test happen in the main daemon loop, in order to reject additional connections without forking additional processes (otherwise a denial-of-service attack could cause a vast number or processes to be created). While the daemon is doing this processing, it cannot accept any other incoming connections.

smtp_accept_queue Type: *integer* Default: *0*

If the number of simultaneous incoming SMTP calls handled via the listening daemon exceeds this value, messages received by SMTP are just placed on the queue; 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_domains**, 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 queue, but no delivery processes are 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, and on dial-in client systems it should probably be set to zero (that is, disabled).

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. The specified hosts are not restricted to this number of connections; the option specifies a minimum number of connection slots for them, not a maximum. It is a guarantee that that group of hosts can always get at least **smtp_accept_reserve** connections.

For example, if **smtp_accept_max** is set to 50 and **smtp_accept_reserve** is set to 5, once there are 45 active connections (from any hosts), new connections are accepted only from hosts listed in **smtp_reserve_hosts**. See also **smtp_accept_max_per_host**.

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 \
  $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: *20*

This option 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 some circumstances such connection attempts have been observed to time out instead. For large systems it is probably a good idea to increase the value (to 50, say). It also gives some protection against denial-of-service attacks by SYN flooding.

smtp_enforce_sync Type: *boolean* Default: *true*

The SMTP protocol specification requires the client to wait for a response from the server at certain points in the dialogue. Without PIPELINING these synchronization points are after every command; with PIPELINING they are fewer, but they still exist. Some spamming sites send out a complete set of SMTP commands without waiting for any response. Exim protects against this by rejecting a message if the client has sent further input when it should not have. The error response '554 SMTP synchronization error' is sent, and the connection is dropped. The check can be disabled by setting **smtp_enforce_sync** false.

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 chapter 37). 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, but Exim does not wait for it to complete. Consequently, its status cannot be checked. If the command cannot be run, a line is written to the panic log, but the ETRN caller still receives a 250 success response. Exim is normally running under its own uid when receiving SMTP, so it is not possible for it to change the uid before running the command.

smtp_etrn_serialize Type: *boolean* Default: *true*

When this option is set, it prevents the simultaneous execution of more than one identical command as a result of ETRN in an SMTP connection. See section 42.9 for details.

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**. If **smtp_reserve_hosts** is not set, no incoming SMTP calls are accepted when the load is over the limit. The option has no effect on ancient operating systems on which Exim cannot determine the load average. See also **deliver_queue_load_max** and **queue_only_load**.

smtp_max_unknown_commands Type: *integer* Default: *3*

If there are too many unrecognized commands in an incoming SMTP session, an Exim server drops the connection. This is a defence against some kinds of abuse that subvert web servers into making connections to SMTP ports; in these circumstances, a number of non-SMTP lines are sent first.

smtp_ratelimit_hosts Type: *host list†* Default: *unset*

Some sites find it helpful to be able to limit the rate at which certain hosts can send them messages, and the rate at which an individual message can specify recipients. When a host matches **smtp_ratelimit_hosts**, the values of **smtp_ratelimit_mail** and **smtp_ratelimit_rcpt** are used to control the rate of acceptance of MAIL and RCPT commands in a single SMTP session, respectively. Each option, if set, must contain a set of four comma-separated values:

- A threshold, before which there is no rate limiting.
- An initial time delay. Unlike other times in Exim, numbers with decimal fractional parts are allowed here.
- A factor by which to increase the delay each time.
- A maximum value for the delay. This should normally be less than 5 minutes, because after that time, the client is liable to timeout the SMTP command.

For example, these settings have been used successfully at the site which first suggested this feature, for controlling mail from their customers:

```
smtp_ratelimit_mail = 2,0.5s,1.05,4m
smtp_ratelimit_rcpt = 4,0.25s,1.015,4m
```

The first setting specifies delays that are applied to MAIL commands after two have been received over a single connection. The initial delay is 0.5 seconds, increasing by a factor of 1.05 each time. The second setting applies delays to RCPT commands when more than four occur in a single message.

It is also possible to configure delays explicitly in ACLs. See section 37.11 for details.

smtp_ratelimit_mail Type: *string* Default: *unset*

See **smtp_ratelimit_hosts** above.

smtp_ratelimit_rcpt Type: *string* Default: *unset*

See **smtp_ratelimit_hosts** above.

smtp_receive_timeout Type: *time* Default: *5m*

This sets a timeout value for SMTP reception. It applies to all forms of SMTP input, including batch SMTP. 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. A line is written to the log containing one of the following messages:

```
SMTP command timeout on connection from...
SMTP data timeout on connection from...
```

The former means that Exim was expecting to read an SMTP command; the latter means that it was in the DATA phase, reading the contents of a message.

The value set by this option can be overridden by the **-os** command-line option. A setting of zero time disables the timeout, but this should never be used for SMTP over TCP/IP. (It can be useful in some cases of local input using **-bs** or **-bS**.) For non-SMTP input, the reception timeout is controlled by **receive_timeout** and **-or**.

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_return_error_details Type: *boolean* Default: *false*

In the default state, Exim uses bland messages such as 'Administrative prohibition' when it rejects SMTP commands for policy reasons. Many sysadmins like this because it gives away little information to spammers. However, some other syadmins who are applying strict checking policies want to give out much fuller information about failures. Setting **smtp_return_error_details** true causes Exim to be more forthcoming. For example, instead of 'Administrative prohibition', it might give:

```
550-Rejected after DATA: '>' missing at end of address:
550 failing address in "From" header is: <user@dom.ain
```

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 sixth 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 be automatically deleted.

When **split_spool_directory** is set, the behaviour of queue runner processes changes. Instead of creating a list of all messages in the queue, and then trying to deliver each one in turn, it constructs a list of those in one sub-directory and tries to deliver them, before moving on to the next sub-directory. The sub-directories are processed in a random order. This spreads out the scanning of the input directories, and uses less memory. It is particularly beneficial when there are lots of messages on the queue. However, if **queue_run_in_order** is set, none of this new processing happens. The entire queue has to be scanned and sorted before any deliveries can start.

spool_directory Type: *string†* Default: *set at compile time*

This defines the directory in which Exim keeps its spool, that is, the messages it is waiting to deliver. 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.

By using this option to override the compiled-in path, it is possible to run tests of Exim without using the standard spool.

strip_excess_angle_brackets Type: *boolean* Default: *false*

If this option is set, 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.

syslog_facility Type: *string* Default: *unset*

This option sets the syslog ‘facility’ name, used when Exim is logging to syslog. The value must be one of the strings ‘mail’, ‘user’, ‘news’, ‘uucp’, ‘daemon’, or ‘local*x*’ where *x* is a digit between 0 and 7. If this option is unset, ‘mail’ is used. See chapter 44 for details of Exim’s logging.

syslog_processname Type: *string* Default: `exim`

This option sets the syslog ‘ident’ name, used when Exim is logging to syslog. The value must be no longer than 32 characters. See chapter 44 for details of Exim’s logging.

syslog_timestamp Type: *boolean* Default: *true*

If **syslog_timestamp** is set false, the timestamps on Exim’s log lines are omitted when these lines are sent to syslog. See chapter 44 for details of Exim’s logging.

system_filter Type: *string*† Default: *unset*

This option specifies a filter file which is applied to all messages at the start of each delivery attempt, before any routing is done. This is called the ‘system message filter’. If the filter generates any deliveries to files or pipes, or any new mail messages, the appropriate **system_filter_..._transport** option(s) must be set, to define which transports are to be used. Details of this facility are given in chapter 39.

system_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. During the delivery, the variable **$address_file** contains the path name.

system_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 specifies a path not ending in ‘/’. During the delivery, the variable **$address_file** contains the path name.

system_filter_group Type: *string* Default: *unset*

This option is used only when **system_filter_user** is also set. It sets the gid under which the system filter is run, overriding any gid that is associated with the user. The value may be numerical or symbolic.

system_filter_pipe_transport Type: *string*† Default: *unset*

This specifies the transport driver that is to be used when a **pipe** command is used in a system filter. During the delivery, the variable **$address_pipe** contains the pipe command.

system_filter_reply_transport Type: *string*† Default: *unset*

This specifies the transport driver that is to be used when a **mail** command is used in a system filter.

system_filter_user Type: *string* Default: *unset*

If this option is not set, the system filter is run in the main Exim delivery process, as root. When the option is set, the system filter runs in a separate process, as the given user. Unless the string consists entirely of digits, it is looked up in the password data. Failure to find the named user causes a configuration error. The gid is either taken from the password data, or specified by **system_filter_group**. When the uid is specified numerically, **system_filter_group** is required to be set.

If the system filter generates any pipe, file, or reply deliveries, the uid under which the filter is run is used when transporting them, unless a transport option overrides. Normally you should set **system_filter_user** if your system filter generates these kinds of delivery.

timeout_frozen_after Type: *time* Default: *0s*

If **timeout_frozen_after** is set to a time greater than zero, a frozen message of any kind that has been on the queue for longer than the given time is automatically cancelled at the next queue run. If it is a bounce message, it is just discarded; otherwise, a bounce is sent to the sender, in a similar manner to cancellation by the **-Mg** command line option. If you want to timeout frozen bounce messages earlier than other kinds of frozen message, see **ignore_bounce_errors_after**.

timezone Type: *string* Default: *unset*

The value of **timezone** is used to set the environment variable TZ while running Exim (if it is different on entry). This ensures that all timestamps created by Exim are in the required timezone. If you want all your timestamps to be in UTC (aka GMT) you should set

```
timezone = UTC
```

The default value is taken from TIMEZONE_DEFAULT in **Local/Makefile**, or, if that is not set, from the value of the TZ environment variable when Exim is built. If **timezone** is set to the empty string, either at build or run time, any existing TZ variable is removed from the environment when Exim runs. This is appropriate behaviour for obtaining wall-clock time on some, but unfortunately not all, operating systems.

tls_advertise_hosts Type: *host list*† Default: *unset*

When Exim is built with support for TLS encrypted connections, the availability of the STARTTLS command to set up an encrypted session is advertised in response to EHLO only to those client hosts that match this option. See chapter 36 for details of Exim's support for TLS.

tls_certificate Type: *string*† Default: *unset*

The value of this option is expanded, and must then be the absolute path to a file which contains the server's certificates. The server's private key is also assumed to be in this file if **tls_privatekey** is unset. See chapter 36 for further details.

tls_dhparam Type: *string*† Default: *unset*

The value of this option is expanded, and must then be the absolute path to a file which contains the server's DH parameter values. This is used only for OpenSSL. When Exim is linked with GnuTLS, this option is ignored. See section 36.1 for further details.

tls_privatekey Type: *string*† Default: *unset*

The value of this option is expanded, and must then be the absolute path to a file which contains the server's private key. If this option is unset, the private key is assumed to be in the same file as the server's certificates. See chapter 36 for further details.

tls_try_verify_hosts Type: *host list*† Default: *unset*

See **tls_verify_hosts** below.

tls_verify_certificates Type: *string*† Default: *unset*

The value of this option is expanded, and must then be the absolute path to a file or a directory containing permitted certificates for clients that match **tls_verify_hosts** or **tls_try_verify_hosts**.

tls_verify_hosts Type: *host list*† Default: *unset*

This option, along with **tls_try_verify_hosts**, controls the checking of certificates from clients. The expected certificates are defined by **tls_verify_certificates**, which must be set. A configuration error occurs if either **tls_verify_hosts** or **tls_try_verify_hosts** is set and **tls_verify_certificates** is not set.

Any client that matches **tls_verify_hosts** is constrained by **tls_verify_certificates**. The client must present one of the listed certificates. If it does not, the connection is aborted.

A weaker form of checking is provided by **tls_try_verify_hosts**. If a client matches this option (but not **tls_verify_hosts**), Exim requests a certificate and checks it against **tls_verify_certificates**, but does not abort the connection if there is no certificate or if it does not match. This state can be detected in an ACL, which makes it possible to implement policies such as 'accept for relay only if a verified certificate has been received, but accept for local delivery if encrypted, even without a verified certificate'.

Client hosts that match neither of these lists are not asked to present certificates.

trusted_groups Type: *string list* Default: *unset*

If this option is set, any process that is running in one of the listed groups, or which has one of them as a supplementary group, is trusted. See section 5.2 for details of what trusted callers are permitted to do. If neither **trusted_groups** nor **trusted_users** is set, only root and the Exim user are trusted.

trusted_users Type: *string list* Default: *unset*

If this option is set, any process that is running as one of the listed users is trusted. See section 5.2 for details of what trusted callers are permitted to do. If neither **trusted_groups** nor **trusted_users** is set, only root and the Exim user are trusted.

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**.

untrusted_set_sender Type: *address list*† Default: *unset*

When an untrusted user submits a message to Exim using the standard input, Exim normally creates an envelope sender address from the user's login and the default qualification domain. Data from the **-f** option (for setting envelope senders on non-SMTP messages) or the SMTP MAIL command (if **-bs** or **-bS** is used) is ignored.

However, untrusted users are permitted to set an empty envelope sender address, to declare that a message should never generate any bounces. For example:

```
exim -f '<>' user@domain.example
```

The **untrusted_set_sender** option allows you to permit untrusted users to set other envelope sender addresses in a controlled way. When it is set, untrusted users are allowed to set envelope sender addresses that match any of the patterns in the list. Like all address lists, the string is expanded. The identity of the user is in **$sender_ident**, so you can, for example, restrict users to setting senders that start with their login ids by a setting like this:

```
untrusted_set_sender = ^$sender_ident-
```

If you want to allow untrusted users to set envelope sender addresses without restriction, you can use

```
untrusted_set_sender = *
```

The **untrusted_set_sender** option applies to all forms of local input, but only to the setting of the envelope sender. It does not permit untrusted users to use the other options which trusted user can use to override message parameters. Furthermore, it does not stop Exim from removing an existing *Sender:* header in the message, or from adding a *Sender:* header if necessary. See **local_sender_retain** and **local_from_check** for ways of overriding these actions. The handling of the *Sender:* header is also described in section 43.12.

The log line for a message's arrival shows the envelope sender following '<='. For local messages, the user's login always follows, after 'U='. In **-bp** displays, and in the Exim monitor, if an untrusted user sets an envelope sender address, the user's login is shown in parentheses after the sender address.

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**. 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.

warn_message_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 40. See also **bounce_message_file**.

14. Generic options for routers

This chapter describes the generic options that apply to all routers, identifying those that are preconditions. For a general description of how a router operates, see sections 3.8 and 3.9. The second of these sections specifies the order in which the preconditions are tested. The order of expansion of the options that provide data for a transport is: **errors_to**, **headers_add**, **headers_remove**, **transport**.

address_data Type: *string*† Default: *unset*

The string is expanded just before the router is run, that is, after all the precondition tests have succeeded. If the expansion is forced to fail, the router declines. Other expansion failures cause delivery of the address to be deferred.

When the expansion succeeds, the value is retained with the address, and can be accessed using the variable **$address_data** in the current router, subsequent routers, and the eventual transport.

Warning: if the current or any subsequent router is a **redirect** router that runs a user's filter file, the contents of **$address_data** are accessible in the filter. This is not normally a problem, because such data is usually either not confidential or it 'belongs' to the current user, but if you do put confidential data into **$address_data** you need to remember this point.

Even if the router declines or passes, the value of **$address_data** remains with the address, though it can be changed by another **address_data** setting on a subsequent router. If a router generates child addresses, the value of **$address_data** propagates to them. This also applies to the special kind of 'child' that is generated by a router with the **unseen** option.

The idea of **address_data** is that you can use it to look up a lot of data for the address once, and then pick out parts of the data later. For example, you could use a single LDAP lookup to return a string of the form

```
uid=1234 gid=5678 mailbox=/mail/xyz forward=/home/xyz/.forward
```

In the transport you could pick out the mailbox by a setting such as

```
file = ${extract{mailbox}{$address_data}}
```

This makes the configuration file less messy, and also reduces the number of lookups. (Exim does cache the most recent lookup, but there may be several addresses in a message which cause lookups to occur.)

The **address_data** facility is also useful as a means of passing information from one router to another, and from a router to a transport. In addition, if **address_data** is set by a router when verifying an address from an ACL, its value is available for use in the rest of the ACL statement.

address_test Type: *boolean (precondition)* Default: *true*

If this option is set false, the router is skipped when routing is being tested by means of the **-bt** command line option. This can be a convenience when your first router sends messages to to an external scanner, because it saves you having to set the 'already scanned' indicator when testing real address routing.

cannot_route_message Type: *string*† Default: *unset*

This option specifies a text message that is used when an address cannot be routed because Exim has run out of routers. The default message is 'Unrouteable address'. This option is useful only on routers that have **more** set false, or on the very last router in a configuration, because the value that is used is taken from the last router that inspects an address. For example, using the default configuration, you could put:

```
cannot_route_message = Remote domain not found in DNS
```

on the first (**dnslookup**) router, and

```
cannot_route_message = Unknown local user
```

on the final router that checks for local users. If string expansion fails, the default message is used; unless the failure was explicitly forced, a message is written to the main and panic logs.

caseful_local_part Type: *boolean* Default: *false*

By default, routers handle the local parts of addresses in a case-insensitive manner, though the actual case is preserved for transmission with the message. If you want the case of letters to be significant in a router, you must set this option true. For individual router options that contain address or local part lists (for example, **local_parts**), case-sensitive matching can be turned on by '+caseful' as a list item. See section 10.14 for more details.

check_local_user Type: *boolean (precondition)* Default: *false*

When this option is true, Exim checks that the local part of the recipient address (with affixes removed if relevant) is the name of an account on the local system. The check is done by calling the *getpwnam()* function rather than trying to read **/etc/passwd** directly. This means that other methods of holding password data (such as NIS) are supported. If the local part is a local user, **$home** is set from the password data, and can be tested in other preconditions that are evaluated after this one (the order of evaluation is given in section 3.9). However, the value of **$home** can be overridden by **router_home_directory**. If the local part is not a local user, the router is skipped.

If you want to check that the local part is either the name of a local user or matches something else, you cannot combine **check_local_user** with a setting of **local_parts**, because that specifies the logical *and* of the two conditions. However, you can use a **passwd** lookup in a **local_parts** setting to achieve this. For example:

```
local_parts = passwd;$local_part : lsearch;/etc/other/users
```

Note, however, that the side effects of **check_local_user** (such as setting up a home directory) do not occur when a **passwd** lookup is used in a **local_parts** (or any other) precondition.

condition Type: *string*† *(precondition)* Default: *unset*

This option specifies a general precondition test that has to succeed for the router 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 router is skipped, and the address is offered to the next one. This provides a means of applying special-purpose conditions to the running of routers.

If the expansion fails (other than forced failure) delivery is deferred. Some of the other options below are common special cases that could in fact be specified using **condition**. Note that **condition** is the last precondition to be evaluated (see section 3.9).

debug_print Type: *string*† Default: *unset*

If this option is set and debugging is enabled (see the **-d** command line option), 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 router configurations. For example, if a **condition** option appears not to be working, **debug_print** can be used to output the variables it references. The output happens after checks for **domains**, **local_parts**, and **check_local_user** but before any other preconditions are tested. A newline is added to the text if it does not end with one.

disable_logging Type: *boolean* Default: *false*

If this option is set true, nothing is logged for any routing errors. You should not set this option unless you really, really know what you are doing.

domains Type: *domain list† (precondition)* Default: *unset*

If this option is set, the router is skipped unless the current domain matches the list. If the match is achieved by means of a file lookup, the data that the lookup returned for the domain is placed in **$domain_data** for use in string expansions of the driver's private options. See section 3.9 for a list of the order in which preconditions are evaluated.

driver Type: *string* Default: *unset*

This option must always be set. It specifies which of the available routers is to be used.

errors_to Type: *string†* Default: *unset*

If a router successfully handles an address, it may queue the address for delivery or it may generate child addresses. In both cases, if there is a delivery problem during later processing, the resulting bounce message is sent to the address that results from expanding this string, provided that the address verifies successfully. **errors_to** is expanded before **headers_add**, **headers_remove**, and **transport**.

If the option is unset, of the expansion is forced to fail, or the result of the expansion fails to verify, the errors address associated with the incoming address is used. At top level, this is the envelope sender. A non-forced expansion failure causes delivery to be deferred.

If an address for which **errors_to** has been set ends up being delivered over SMTP, the envelope sender for that delivery is the **errors_to** value, so that any bounces that are generated by other MTAs on the delivery route are also sent there. The most common use of **errors_to** is probably to direct mailing list bounces to the manager of the list, as described in section 41.2.

The **errors_to** setting associated with an address can be overridden if it subsequently passes through other routers that have their own **errors_to** settings, or if it is delivered by a transport with a **return_path** setting.

You can set **errors_to** to the empty string by either of these settings:

```
errors_to =
errors_to = ""
```

An expansion item that yields an empty string has the same effect. If you do this, a locally detected delivery error for addresses processed by this router no longer gives rise to a bounce message; the error is discarded. If the address is delivered to a remote host, the return path is set to **<>**, unless overridden by the **return_path** option on the transport.

If for some reason you want to discard local errors, but use a non-empty MAIL command for remote delivery, you can preserve the original return path in **$address_data** in the router, and reinstate it in the transport by setting **return_path**.

expn Type: *boolean (precondition)* Default: *true*

If this option is turned off, the router 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 router for users' **.forward** files, while leaving it on for the system alias file. See section 3.9 for a list of the order in which preconditions are evaluated.

The use of the SMTP EXPN command is controlled by an ACL (see chapter 37). When Exim is running an EXPN command, it is similar to testing an address with **-bt**. Compare VRFY, whose counterpart is **-bv**.

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 router when verifying a recipient, verification fails.

fail_verify_sender Type: *boolean* Default: *false*

If this option is true and an address is accepted by this router when verifying a sender, verification fails.

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 router 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. If **hosts_randomize** is set on the transport, the order of the list is randomized for each use. See the **fallback_hosts** option of the **smtp** transport for further details.

group Type: *string*† Default: *see below*

When a router queues an address for a transport, and the transport does not specify a group, the group given here is used when running the delivery process. The default is unset, but if **check_local_user** is set, the default is taken from the password information. See also **initgroups** and **user** and the discussion in chapter 22.

headers_add Type: *string*† Default: *unset*

This option specifies a string of text that is expanded at routing time, and associated with any addresses that are processed by the router when delivering a message. This option has no effect when an address is just being verified.

The **headers_add** option is expanded after **errors_to**, but before **headers_remove** and **transport**. 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. The expanded string must be in the form of one or more RFC 2822 header lines, separated by newlines (coded as '\n'). For example:

```
headers_add = X-added-header: added by $primary_hostname\n\
              X-added-second: another added header line
```

Exim does not check the syntax of these added header lines, except that a newline is supplied at the end if one is not present. If an address passes through several routers as a result of aliasing or forwarding operations, any **headers_add** or **headers_remove** specifications are cumulative. This does not apply for multiple routers that result from the use of 'unseen'.

At transport time, all the original headers listed in **headers_remove** are removed. If there are multiple instances of any listed header, they are all removed. Then the new headers specified by **headers_add** are added, in the order in which they were attached to the address. Finally, any additional headers specified by the transport are added. It is not possible to remove headers added to an address by **headers_add**.

Because the addition does not happen until transport time, header lines that are added by **headers_add** are not accessible by means of the **$header_xxx** expansion syntax. Conversely, header lines that are removed by **headers_remove** remain visible.

Addresses with different **headers_add** or **headers_remove** settings cannot be delivered together in a batch. The **headers_add** option cannot be used for a **redirect** router that has the **one_time** option set.

headers_remove Type: *string*† Default: *unset*

The string is expanded at routing time and is then associated with any addresses that are processed by the router when delivering a message. This option has no effect when an address is just being verified. The **headers_remove** option is expanded after **errors_to** and **headers_add**, but before **transport**. 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:

```
headers_remove = return-receipt-to:acknowledge-to
```

It is used at transport time as described under **headers_add** above. The **headers_remove** option cannot be used for a **redirect** router that has the **one_time** option set.

ignore_target_hosts Type: *host list†* Default: *unset*

Although this option is a host list, it should normally contain IP address entries rather than names. If any host that is looked up by the router has an IP address that matches an item in this list, Exim behaves as if that IP address did not exist. This option allows you to cope with rogue DNS entries like

```
remote.domain.example  A  127.0.0.1
```

by setting

```
ignore_target_hosts = 127.0.0.1
```

on the relevant router. An attempt to mail to such a domain then provokes the 'unrouteable domain' error, and an attempt to verify an address in the domain fails. This option may also be useful for ignoring link-local and site-local IPv6 addresses. Because, like all host lists, the value of **ignore_target_hosts** is expanded before use as a list, it is possible to make it dependent on the domain that is being routed.

initgroups Type: *boolean* Default: *false*

If the router queues an address for a transport, and this option is true, and the uid supplied by the router is not overridden by the transport, 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 22.

local_part_prefix Type: *string list (precondition)* Default: *unset*

If this option is set, the router is skipped unless the local part starts with one of the given strings, or **local_part_prefix_optional** is true. See section 3.9 for a list of the order in which preconditions are evaluated.

The list is scanned from left to right, and the first prefix that matches is used. 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 section 41.7.

During the testing of the **local_parts** option, and while the router is running, the prefix is removed from the local part, and is available in the expansion variable **$local_part_prefix**. If the router accepts the address, 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 router like this one immediately before the router that handles **.forward** files:

```
real_localuser:
  driver = accept
  local_part_prefix = real-
  check_local_user
  transport = local_delivery
```

If both **local_part_prefix** and **local_part_suffix** are set for a router, 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 router. Different separator characters must be used to avoid ambiguity.

local_part_prefix_optional Type: *boolean* Default: *false*

See **local_part_prefix** above.

local_part_suffix Type: *string list (precondition)* Default: *unset*

This option operates in the same way as **local_part_prefix**, except that the local part must end (rather than start) with the given string, the **local_part_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**.

local_part_suffix_optional Type: *boolean* Default: *false*

See **local_part_suffix** above.

local_parts Type: *local part list† (precondition)* Default: *unset*

The router is run only if the local part of the address matches the list. See section 3.9 for a list of the order in which preconditions are evaluated, and section 10.15 for a discussion of local part lists. 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, the data that the lookup returned for the local part is placed in the variable **$local_part_data** for use in expansions of the router'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:
  driver = redirect
  local_parts = postmaster
  data = postmaster@real.domain.example
```

log_as_local Type: *boolean* Default: *see below*

Exim has two logging styles for delivery, the idea being to make local deliveries stand out more visibly from remote ones. In the 'local' style, the recipient address is given just as the local part, without a domain. The use of this style is controlled by this option. It defaults to true for the **accept** router, and false for all the others.

more Type: *boolean†* Default: *true*

The result of string expansion for this option must be a valid boolean value, that is, one of the strings 'yes', 'no', 'true', or 'false'. Any other result causes an error, and delivery is deferred. If the expansion is forced to fail, the default value for the option (true) is used. Other failures cause delivery to be deferred.

If this option is set false, and the router is run, but declines to handle the address, no further routers are tried, routing fails, and the address is bounced. However, if the router explicitly passes an address to the following router by means of the setting

```
self = pass
```

or otherwise, the setting of **more** is ignored. Also, the setting of **more** does not affect the behaviour if one of the precondition tests fails. In that case, the address is always passed to the next router.

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 passed on to the next router, overriding **no_more**. 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.

pass_router Type: *string* Default: *unset*

When a router returns ‘pass’, the address is normally handed on to the next router in sequence. This can be changed by setting **pass_router** to the name of another router. However (unlike **redirect_router**) the named router must be below the current router, to avoid loops. Note that this option applies only to the special case of ‘pass’. It does not apply when a router returns ‘decline’.

redirect_router Type: *string* Default: *unset*

Sometimes an administrator knows that it is pointless to reprocess addresses generated from alias or forward files with the same router again. For example, if an alias file translates real names into login ids there is no point searching the alias file a second time, especially if it is a large file.

The **redirect_router** option can be set to the name of any router instance. It causes the routing of any generated addresses to start at the named router instead of at the first router. This option has no effect if the router in which it is set does not generate new addresses.

require_files Type: *string list† (precondition)* Default: *unset*

This option provides a general mechanism for predicating the running of a router on the existence or non-existence of certain files or directories. Before running a router, as one of its precondition tests, Exim works its way through the **require_files** list, expanding each item separately.

Because the list is split before expansion, any colons in expansion items must be doubled, or the facility for using a different list separator must be used. If any expansion is forced to fail, the item is ignored. Other expansion failures cause routing of the address to be deferred.

If any expanded 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 router is skipped if any paths not preceded by ‘!’ do not exist, or if any paths preceded by ‘!’ do exist.

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.

This option is checked after the **domains**, **local_parts**, and **senders** options, so you cannot use it to check for the existence of a file in which to look up up a domain, local part, or sender. (See section 3.9 for a full list of the order in which preconditions are evaluated.) However, as these options are all expanded, you can use the **exists** expansion condition to make such tests. The **require_files** option is intended for checking files that the router may be going to use internally, or which are needed by a transport (for example **.procmailrc**).

During delivery, the *stat()* function is run as root, but there is a facility for checking the accessibility of a file by another user. 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 checked for subsequent files in the list. If no group is specified but the user is specified symbolically, the gid associated with the uid is used. For example:

```
require_files = mail:/some/file
require_files = $local_part:$home/.procmailrc
```

If a user or group name in a **require_files** list does not exist, the **require_files** condition fails.

Exim performs the check by scanning along the components of the file path, and checking the access for the given uid and gid. It checks for ‘x’ access on directories, and ‘r’ access on the final file. Note that this means that file access control lists, if the operating system has them, are ignored.

Warning: When the router is being run to verify addresses for an incoming SMTP message, Exim is not running as root, but under its own uid. This may affect the result of a **require_files** check. In particular, *stat()* may yield the error EACCES (‘Permission denied’). This means that the Exim 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 routing 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, the EACCES error is treated as if the file did not exist. For example:

```
require_files = +/some/file
```

If the router is not an essential part of verification (for example, it handles users' **.forward** files), another solution is to set the **verify** option false so that the router is skipped when verifying.

retry_use_local_part Type: *boolean* Default: *see below*

When a delivery suffers a temporary routing failure, a retry record is created in Exim's hints database. For addresses whose routing depends only on the domain, the key for the retry record should not involve the local part, but for other addresses, both the domain and the local part should be included. Usually, remote routing is of the former kind, and local routing is of the latter kind.

This option controls whether the local part is used to form the key for retry hints for addresses that suffer temporary errors while being handled by this router. The default value is true for any router that has **check_local_user** set, and false otherwise. Note that this option does not apply to hints keys for transport delays; they are controlled by a generic transport option of the same name.

router_home_directory Type: *string*† Default: *unset*

This option sets a home directory for use while the router is running. (Compare **transport_ home_directory**, which sets a home directory for later transporting.) In particular, if used on a **redirect** router, this option sets a value for **$home** while a filter is running. The value is expanded; forced expansion failure causes the option to be ignored – other failures cause the router to defer.

Expansion of **router_home_directory** happens immediately after the **check_local_user** test (if configured), before any further expansions take place. (See section 3.9 for a list of the order in which preconditions are evaluated.) While the router is running, **router_home_directory** overrides the value of **$home** that came from **check_local_user**.

When a router accepts an address and routes it to a transport (including the cases when a redirect router generates a pipe, file, or autoreply delivery), the home directory setting for the transport is taken from the first of these values that is set:

- The **home_directory** option on the transport;
- The **transport_home_directory** option on the router;
- The password data if **check_local_user** is set on the router;
- The **router_home_directory** option on the router.

In other words, **router_home_directory** overrides the password data for the router, but not for the transport.

self Type: *string* Default: *freeze*

This option applies to those routers which use the address to find a list of remote hosts. Currently, these are the **dnslookup**, **ipliteral**, and **manualroute** routers. Usually such routers are configured to send the message to a remote host via an **smtp** transport. The **self** option specifies what happens when the first host on the list turns out to be the local host (this is checked by comparing IP addresses), or a host whose name matches **hosts_treat_as_local**.

Normally this situation indicates either an error in Exim's configuration (for example, the router should be configured not to process this domain), or an error in the DNS (for example, the MX should not point to this host). For this reason, the default action is to log the incident, defer the address, and freeze the message. The following alternatives are provided for use in special cases:

- **defer**
  Delivery of the message is tried again later, but the message is not frozen.
- **reroute:** <*domain*>
  The domain is changed to the given domain, and the address is passed back to be reprocessed by the routers. No rewriting of headers takes place. This behaviour is essentially a redirection.

- **reroute: rewrite:** <*domain*>
  The domain is changed to the given domain, and the address is passed back to be reprocessed by the routers. Any headers that contain the original domain are rewritten.

- **pass**
  The router passes the address to the next router, or to the router named in the **pass_router** option if it is set. This overrides **no_more**.

  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. The combination

  ```
  self = pass
  no_more
  ```

  ensures that only those addresses that routed to the local host are passed on. Without **no_more**, addresses that were declined for other reasons would also be passed to the next router.

- **fail**
  Delivery fails and an error report is generated.

- **send**
  The anomaly is ignored and the address is queued for the transport. This setting should be used with extreme caution. For an **smtp** transport, 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.

senders Type: *address list*† *(precondition)* Default: *unset*

If this option is set, the router is skipped unless the message's sender address matches something on the list. See section 3.9 for a list of the order in which preconditions are evaluated.

There are issues concerning verification when the running of routers is dependent on the sender. When Exim is verifying the address in an **errors_to** setting, it sets the sender to the null string. When 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.

translate_ip_address Type: *string*† Default: *unset*

There exist some rare networking situations (for example, packet radio) where it is helpful to be able to translate IP addresses generated by normal routing mechanisms into other IP addresses, thus performing a kind of manual IP routing. This should be done only if the normal IP routing of the TCP/IP stack is inadequate or broken. Because this is an extremely uncommon requirement, the code to support this option is not included in the Exim binary unless SUPPORT_TRANSLATE_IP_ADDRESS=yes is set in **Local/Makefile**.

The **translate_ip_address** string is expanded for every IP address generated by the router, with the generated address set in **$host_address**. If the expansion is forced to fail, no action is taken. If it returns an IP address, that replaces the original address; otherwise the result is assumed to be a host name – this is looked up using *gethostbyname()* (or *getipnodebyname()* when available) to produce one or more replacement IP addresses. For example, to subvert all IP addresses in some specific networks, this could be added to a router:

```
translate_ip_address = \
  ${lookup{${mask:$host_address/26}}lsearch{/some/file}{$value}fail}
```

The file would contain lines like

```
10.2.3.128/26    some.host
10.8.4.34/26     10.44.8.15
```

You should not make use of this facility unless you really understand what you are doing.

transport Type: *string*† Default: *unset*

This option specifies the transport to be used when a router accepts an address and sets it up for delivery. A transport is never needed if a router is used only for verification. The value of the option is expanded at routing time, after the expansion of **errors_to**, **headers_add**, and **headers_remove**, and result must be the name of one of the configured transports. If it is not, delivery is deferred.

The **transport** option is not used by the **redirect** router, but it does have some private options that set up transports for pipe and file deliveries (see chapter 21).

transport_current_directory Type: *string*† Default: *unset*

This option associates a current directory with any address that is routed to a local transport. This can happen either because a transport is explicitly configured for the router, or because it generates a delivery to a file or a pipe. During the delivery process (that is, at transport time), this option string is expanded and is set as the current directory, unless overridden by a setting on the transport. See chapter 22 for details of the local delivery environment.

transport_home_directory Type: *string*† Default: *see below*

This option associates a home directory with any address that is routed to a local transport. This can happen either because a transport is explicitly configured for the router, or because it generates a delivery to a file or a pipe. During the delivery process (that is, at transport time), the option string is expanded and is set as the home directory, unless overridden by a setting of **home_directory** on the transport.

If the transport does not specify a home directory, and **transport_home_directory** is not set for the router, the home directory for the tranport is taken from the password data if **check_local_user** is set for the router. Otherwise it is taken from **router_home_directory** if that option is set; if not, no home directory is set for the transport.

See chapter 22 for further details of the local delivery environment.

unseen Type: *boolean*† Default: *false*

The result of string expansion for this option must be a valid boolean value, that is, one of the strings ‘yes’, ‘no’, ‘true’, or ‘false’. Any other result causes an error, and delivery is deferred. If the expansion is forced to fail, the default value for the option (false) is used. Other failures cause delivery to be deferred.

When this option is set true, routing does not cease if the router accepts the address. Instead, a copy of the incoming address is passed to the next router, overriding a false setting of **more**. There is little point in setting **more** false if **unseen** is always true, but it may be useful in cases when the value of **unseen** contains expansion items (and therefore, presumably, is sometimes true and sometimes false).

The **unseen** option can be used to cause copies of messages to be delivered to some other destination, while leaving the normal delivery untouched. The effect is to clone the address before processing one copy of it, so options such as **headers_add** on the current router do not affect the other copy. However, any data that was set by the **address_data** option in the current or previous routers is passed on. Setting this option has a similar effect to the **unseen** command qualifier in filter files.

user Type: *string*† Default: *see below*

When a router queues an address for a transport, and the transport does not specify a user, the user given here is used when running the delivery process. This user is also used by the **redirect** router when running a filter file. The default is unset, except when **check_local_user** is set. In this case, the default is taken from the password information. If the user is specified as a name, and **group** is not set, the group associated with the user is used. See also **initgroups** and **group** and the discussion in chapter 22.

verify Type: *boolean (precondition)* Default: *true*

Setting this option has the effect of setting **verify_sender** and **verify_recipient** to the same value.

verify_only Type: *boolean (precondition)* Default: *false*

If this option is set, the router 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. It can be further restricted to verifying only senders or recipients by means of **verify_sender** and **verify_recipient**.

Warning: When the router is being run to verify addresses for an incoming SMTP message, Exim is not running as root, but under its own uid. If the router accesses any files, you need to make sure that they are accessible to the Exim user or group.

verify_recipient Type: *boolean (precondition)* Default: *true*

If this option is false, the router is skipped when verifying recipient addresses. See section 3.9 for a list of the order in which preconditions are evaluated.

verify_sender Type: *boolean (precondition)* Default: *true*

If this option is false, the router is skipped when verifying sender addresses. See section 3.9 for a list of the order in which preconditions are evaluated.

15. The accept router

The **accept** router has no private options of its own. Unless it is being used purely for verification (see **verify_only**) a transport is required to be defined by the generic **transport** option. If the preconditions that are specified by generic options are met, the router accepts the address and queues it for the given transport. The most common use of this router is for setting up deliveries to local mailboxes. For example:

```
localusers:
  driver = accept
  domains = mydomain.example
  check_local_user
  transport = local_delivery
```

The **domains** condition in this example checks the domain of the address, and **check_local_user** checks that the local part is the login of a local user. When both preconditions are met, the **accept** router runs, and queues the address for the **local_delivery** transport.

16. The dnslookup router

The **dnslookup** router looks up the hosts that handle mail for the given domain in the DNS. A transport must always be set for this router, unless **verify_only** is set.

MX records are looked up first, followed by address records if no MX records are found, unless the domain matches **mx_domains**. MX records of equal priority are sorted by Exim into a random order. 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 looked up as an address record, is the local host, or matches **hosts_treat_as_local**, what happens is controlled by the generic **self** option.

There are a number of private options that can be used to vary the way the DNS lookup is handled.

check_secondary_mx Type: *boolean* Default: *false*

If this option is set, the router declines 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.

mx_domains Type: *domain list†* Default: *unset*

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.example* are known to have MX records, except for those in *discworld.fict.example*, you could use this setting:

```
mx_domains = ! *.discworld.fict.example : *.fict.example
```

This specifies that messages addressed to a domain that matches the list but has no MX record should be bounced immediately instead of being routed using the address record.

qualify_single Type: *boolean* Default: *true*

When this option is true, the resolver option RES_DEFNAMES is set for DNS lookups. Typically, but not standardly, this causes the resolver to qualify single-component names with the default domain. For example, on a machine called *dictionary.ref.example*, the domain *thesaurus* would be changed to *thesaurus.ref.example* inside the resolver. For details of what your resolver actually does, consult your man pages for *resolver* and *resolv.conf*.

rewrite_headers Type: *boolean* Default: *true*

If the domain name in the address that is being processed is not fully qualified, it may be expanded to its full form by a DNS lookup. For example, if an address is specified as *dormouse@teaparty*, the domain might be expanded to *teaparty.wonderland.fict.example*. Domain expansion can also occur as a result of setting the **widen_domains** option. If **rewrite_headers** is true, all occurrences of the abbreviated domain name in any *Bcc:*, *Cc:*, *From:*, *Reply-to:*, *Sender:*, and *To:* header lines of the message are rewritten with the full domain 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, name servers 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 name servers 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.

same_domain_copy_routing Type: *boolean* Default: *false*

Addresses with the same domain are normally routed by the **dnslookup** router to the same list of hosts. However, this cannot be presumed, because the router options and preconditions may refer to the local part of the address. By default, therefore, Exim routes each address in a message independently. DNS servers run caches, so repeated DNS lookups are not normally expensive, and in any case, personal messages rarely have more than a few recipients.

If you are running mailing lists with large numbers of subscribers at the same domain, and you are using a **dnslookup** router which is independent of the local part, you can set **same_domain_copy_routing** to bypass repeated DNS lookups for identical domains in one message. In this case, when **dnslookup** routes an address to a remote transport, any other unrouted addresses in the message that have the same domain are automatically given the same routing without processing them independently. However, this is only done if **headers_add** and **headers_remove** are unset, and the router does not ‘widen’ the domain.

search_parents Type: *boolean* Default: *false*

When this option is true, the resolver option RES_DNSRCH is set for DNS lookups. This is different from the **qualify_single** option in that it applies to domains containing dots. Typically, but not standardly, it causes the resolver to search for the name in the current domain and in parent domains. For example, on a machine in the *fict.example* domain, if looking up *teaparty.wonderland* failed, the resolver would try *teaparty.wonderland.fict.example*. For details of what your resolver actually does, consult your man pages for *resolver* and *resolv.conf*.

Setting this option true can cause problems in domains that have a wildcard MX record, because any domain that does not have its own MX record matches the local wildcard.

widen_domains Type: *string list* Default: *unset*

If a DNS 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.example:ref.example
```

is set and a lookup of *klingon.dictionary* fails, *klingon.dictionary.fict.example* is looked up, and if this fails, *klingon.dictionary.ref.example* is tried. Note that the **qualify_single** and **search_parents** options can cause some widening to be undertaken inside the DNS resolver.

17. The ipliteral router

This router has no private options. Unless it is being used purely for verification (see **verify_only**) a transport is required to be defined by the generic **transport** option. The router accepts the address if its domain part takes the form of an RFC 2822 domain literal, that is, an IP address enclosed in square brackets. For example, this router handles the address

```
root@[192.168.1.1]
```

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.

If you want to use this router, you must also set the main configuration option **allow_domain_literals**. Otherwise, Exim will not recognize the domain literal syntax in addresses.

18. The iplookup router

The **iplookup** router was written to fulfil a specific requirement in Cambridge University. For this reason, it is not included in the binary of Exim by default. If you want to include it, 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. The new address is then passed on to subsequent routers.

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 items 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()* (or *getipnodebyname()* when available) and are tried in order until one responds to the query. If none respond, what happens is controlled by **optional**.

optional Type: *boolean* Default: *false*

If **optional** is true, if no response is obtained from any host, the address is passed to the next router, overriding **no_more**. If **optional** is false, delivery to the 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 declines. 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
```

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.

19. The manualroute router

The **manualroute** router is so-called because it provides a way of manually routing an address according to its domain. It is mainly used when you want to route addresses to remote hosts according to your own rules, bypassing the normal DNS routing that looks up MX records. However, **manualroute** can also route to local transports, a facility that may be useful if you want to save messages for dial-in hosts in local files.

The **manualroute** router compares a list of domain patterns with the domain it is trying to route. If there is no match, the router declines. Each pattern has associated with it a list of hosts and some other optional data, which may include a transport. The combination of a pattern and its data is called a 'routing rule'. For patterns that do not have an associated transport, the generic **transport** option must specify a transport, unless the router is being used purely for verification (see **verify_only**).

In the case of verification, matching the domain pattern is sufficient for the router to accept the address. When actually routing an address for delivery, an address that matches a domain pattern is queued for the associated transport. If the transport is not a local one, a host list must be associated with the pattern; IP addresses are looked up for the hosts, and these are passed to the transport along with the mail address. For local transports, a host list is optional. If it is present, it is passed in **$host** as a single text string.

The list of routing rules can be provided as an inline string in **route_list**, or the data can be obtained by looking up the domain in a file or database by setting **route_data**. Only one of these settings may appear in any one instance of **manualroute**. The format of routing rules is described below, following the list of private options.

19.1 Private options for manualroute

The private options for the **manualroute** router are as follows:

host_find_failed Type: *string* Default: *freeze*

This option controls what happens when **manualroute** tries to find an IP address for a host, and the host does not exist. The option can be set to one of

```
decline
defer
fail
freeze
pass
```

The default assumes that this state is a serious configuration error. The difference between 'pass' and 'decline' is that the former forces the address to be passed to the next router (or the router defined by **pass_router**), overriding **no_more**, whereas the latter passes the address to the next router only if **more** is true.

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.

hosts_randomize Type: *boolean* Default: *false*

If this option is set, the order of the items in a host list in a routing rule is randomized each time the list is used, unless an option in the routing rule overrides (see below). Randomizing the order of a host list can be used to do crude load sharing. However, if more than one mail address is routed by the same router to the same host list, the host lists are considered to be the same (even though they may be randomized into different orders) for the purpose of deciding whether to batch the deliveries into a single SMTP transaction.

When **hosts_randomize** is true, a host list may be split into groups whose order is separately randomized. This makes it possible to set up MX-like behaviour. The boundaries between groups are indicated by an item that is just + in the host list. For example:

```
route_list = * host1:host2:host3:+:host4:host5
```

The order of the first three hosts and the order of the last two hosts is randomized for each use, but the first three always end up before the last two. If **hosts_randomize** is not set, a + item in the list is ignored. If a randomized host list is passed to an **smtp** transport that also has **hosts_randomize set**, the list is not re-randomized.

route_data Type: *string†* Default: *unset*

If this option is set, it must expand to yield the data part of a routing rule. Typically, the expansion string includes a lookup based on the domain. For example:

```
route_data = ${lookup{$domain}dbm{/etc/routes/}}
```

If the expansion is forced to fail, or the result is an empty string, the router declines. Other kinds of expansion failure cause delivery to be deferred.

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.

same_domain_copy_routing Type: *boolean* Default: *false*

Addresses with the same domain are normally routed by the **manualroute** router to the same list of hosts. However, this cannot be presumed, because the router options and preconditions may refer to the local part of the address. By default, therefore, Exim routes each address in a message independently. DNS servers run caches, so repeated DNS lookups are not normally expensive, and in any case, personal messages rarely have more than a few recipients.

If you are running mailing lists with large numbers of subscribers at the same domain, and you are using a **manualroute** router which is independent of the local part, you can set **same_domain_copy_routing** to bypass repeated DNS lookups for identical domains in one message. In this case, when **manualroute** routes an address to a remote transport, any other unrouted addresses in the message that have the same domain are automatically given the same routing without processing them independently. However, this is only done if **headers_add** and **headers_remove** are unset.

19.2 Routing rules in route_list

The value 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> <list of hosts> <options>

The following example contains two rules, each with a simple domain pattern and no options:

```
route_list = \
  dict.ref.example  mail-1.ref.example:mail-2.ref.example ; \
  thes.ref.example  mail-3.ref.example:mail-4.ref.example
```

The three parts of a rule are separated by white space. The pattern and the list of hosts can be enclosed in quotes if necessary, and if they are, the usual quoting rules apply. 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 10.7), 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 list of hosts and then options are then used as described below. If there is no match, the router declines. When **route_list** is set, **route_data** must not be set.

19.3 Routing rules in route_data

The use of **route_list** is convenient when there are only a small number of routing rules. For larger numbers, it is easier to use a file or database to hold the routing information, and use the **route_data** option instead. The value of **route_data** is a list of hosts, followed by (optional) options. Most commonly, **route_data** is set as a string that contains an expansion lookup. For example, suppose we place two routing rules in a file like this:

```
dict.ref.example:  mail-1.ref.example:mail-2.ref.example
thes.ref.example:  mail-3.ref.example:mail-4.ref.example
```

This data can be accessed by setting

```
route_data = ${lookup{$domain}lsearch{/the/file/name}}
```

Failure of the lookup results in an empty string, causing the router to decline. However, you do not have to use a lookup in **route_data**. The only requirement is that the result of expanding the string is a list of hosts, possibly followed by options, separated by white space. The list of hosts must be enclosed in quotes if it contains white space.

19.4 Format of the list of hosts

A list of hosts, whether obtained via **route_data** or **route_list**, is always separately expanded before use. If the expansion fails, the router declines. The result of the expansion must be a colon-separated list of host names and/or IP addresses. IP addresses are not enclosed in brackets.

If the list of hosts was obtained from a **route_list** item, the following variables are set during its expansion:

- If the domain was matched against a regular expression, the numeric variables **$1**, **$2**, etc. may be set.
- **$0** is always set to the entire domain.
- **$1** is also set when partial matching is done in a file lookup.
- If the pattern that matched the domain was a lookup item, the data that was looked up is available in the expansion variable **$value**.

19.5 How the list of hosts is used

When an address is routed to an **smtp** transport by **manualroute**, each of the hosts is tried, in order, when carrying out the SMTP delivery. However, this can be changed by setting the **hosts_randomize** option, either on the router (see section 19.1 above), or on the transport.

19.6 How the options are used

The options are a sequence of words; in practice no more than three are ever present. One of the words can be the name of a transport; this overrides the **transport** option on the router for this particular routing rule only. The other words (if present) control randomization of the list of hosts on a per-rule basis, and how the IP addresses of the hosts are to be found when routing to a remote transport. These options are as follows:

- **randomize**: randomize the order of the hosts in this list, overriding the setting of **hosts_randomize** for this routing rule only.
- **no_randomize**: do not randomize the order of the hosts in this list, overriding the setting of **hosts_randomize** for this routing rule only.

- **byname**: use *getipnodebyname()* (*gethostbyname()* on older systems) to find IP addresses. This function may ultimately cause a DNS lookup, but it may also look in **/etc/hosts** or other sources of information.
- **bydns**: look up address records for the hosts directly in the DNS; fail if no address records are found. If there is a temporary DNS error (such as a timeout), delivery is deferred.

For example:

```
route_list = domain1  host1:host2:host3  randomize bydns;\
             domain2  host4:host5
```

If neither **byname** nor **bydns** is given, Exim behaves as follows: First, a DNS lookup is done. If this yields anything other than HOST_NOT_FOUND, that result is used. Otherwise, Exim goes on to try a call to *getipnodebyname()* or *gethostbyname()*, and the result of the lookup is the result of that call.

Warning: It has been discovered that on some systems, if a DNS lookup called via *getipnodebyname()* times out, HOST_NOT_FOUND is returned instead of TRY_AGAIN. That is why the default action is to try a DNS lookup first. Only if that gives a definite 'no such host' is the local function called.

If no IP address for a host can be found, what happens is controlled by the **host_find_failed** option.

When an address is routed to a local transport, IP addresses are not looked up. The host list is passed to the transport in the **$host** variable.

19.7 Manualroute examples

In some of the examples that follow, the presence of the **remote_smtp** transport, as defined in the default configuration file, is assumed:

- The **manualroute** router can be used to forward all external mail to a *smart host*. If you have set up, in the main part of the configuration, a named domain list that contains your local domains, for example,

  ```
  domainlist local_domains = my.domain.example
  ```

  you can arrange for all other domains to be routed to a smart host by making your first router something like this:

  ```
  smart_route:
    driver = manualroute
    domains = !+local_domains
    transport = remote_smtp
    route_list = * smarthost.ref.example
  ```

  This causes all non-local addresses to be sent to the single host *smarthost.ref.example*. If a colon-separated list of smart hosts is given, they are tried in order (but you can use **hosts_randomize** to vary the order each time). Another way of configuring the same thing is this:

  ```
  smart_route:
    driver = manualroute
    transport = remote_smtp
    route_list = !+local_domains  smarthost.ref.example
  ```

  There is no difference in behaviour between these two routers as they stand. However, they behave differently if **no_more** is added to them. In the first example, the router is skipped if the domain does not match the **domains** precondition; the following router is always tried. If the router runs, it always matches the domain and so can never decline. Therefore, **no_more** would have no effect. In the second case, the router is never skipped; it always runs. However, if it doesn't match the domain, it declines. In this case **no_more** would prevent subsequent routers from running.

- A *mail hub* is a host 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 **manualroute** router is usually used on a mail hub to route incoming messages to the correct hosts. For a small number of domains, the routing can be inline, using the **route_list** option, but for a larger number a file or database lookup is easier to manage.

If the domain names are in fact the names of the machines to which the mail is to be sent by the mail hub, the configuration can be quite simple. For example,

```
hub_route:
  driver = manualroute
  transport = remote_smtp
  route_list = *.rhodes.tvs.example  $domain
```

This configuration routes domains that match `*.rhodes.tvs.example` to hosts whose names are the same as the mail domains. A similar approach can be taken if the host name can be obtained from the domain name by a string manipulation that the expansion facilities can handle. Otherwise, a lookup based on the domain can be used to find the host:

```
through_firewall:
  driver = manualroute
  transport = remote_smtp
  route_data = ${lookup {$domain} cdb {/internal/host/routes}}
```

The result of the lookup must be the name or IP address of the host (or hosts) to which the address is to be routed. If the lookup fails, the route data is empty, causing the router to decline. The address then passes to the next router.

- You can use **manualroute** to deliver messages to pipes or files in batched SMTP format for onward transportation by some other means. This is one way of storing mail for a dial-up host when it is not connected. The route list entry can be as simple as a single domain name in a configuration like this:

  ```
  save_in_file:
    driver = manualroute
    transport = batchsmtp_appendfile
    route_list = saved.domain.example
  ```

  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:

  ```
  save_in_file:
    driver = manualroute
    route_list = \
      *.saved.domain1.example  $domain  batch_appendfile; \
      *.saved.domain2.example  \
        ${lookup{$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 decline to handle the address if the lookup fails.

- Routing mail directly to UUCP software is a specific case of the use of **manualroute** in a gateway to another mail environment. This is an example of one way it can be done:

```
# 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 = manualroute
  route_data = \
    ${lookup{$domain}lsearch{/usr/local/exim/uucphosts}}
```

The file **/usr/local/exim/uucphosts** contains entries like

```
darksite.ethereal.example:           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.example* and the UUCP host name *darksite*.

20. The queryprogram router

The **queryprogram** router routes an address 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 precondition options (**domains**, **local_parts**, etc) to skip this router for most addresses, it could sensibly be used in special cases, even on a busy host. There are the following private options:

command Type: *string†* Default: *unset*

This option must be set. It specifies the command that is to be run. The command is split up into a command name and arguments, and then each is expanded separately (exactly as for a **pipe** transport, described in chapter 28).

command_group Type: *string* Default: *unset*

This option specifies a gid to be set when running the command. It must be set if **command_user** specifies a numerical uid. 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 must be set. It 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: */*

This option specifies an absolute path which is made the current directory before running the command.

timeout Type: *time* Default: *1h*

If the command does not complete within the timeout period, its process group is killed and the message is frozen. A value of zero time specifies no timeout.

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 (case-insensitive):

- *Accept*: routing succeeded; the remaining fields specify what to do (see below).
- *Decline*: the router declines; pass the address to the next router, unless **no_more** is set.
- *Fail*: routing failed; do not pass the address to any more routers. Any subsequent text on the line is an error message. If the router is run as part of address verification during an incoming SMTP message, the message is included in the SMTP response.
- *Defer*: routing could not be completed at this time; try again later. Any subsequent text on the line is an error message which is logged. It is not included in any SMTP response.
- *Freeze*: the same as *defer*, except that the message is frozen.
- *Pass*: pass the address to the next router (or the router specified by **pass_router**), overriding **no_more**.
- *Redirect*: the message is redirected. The remainder of the line is a list of new addresses, which are routed independently, starting with the first router, or the router specified by **redirect_router**, if set.

When the first word is *accept*, the remainder of the line consists of a number of keyed data values, as follows (split into two lines here, to fit on the page):

```
ACCEPT TRANSPORT=<transport> HOSTS=<host list>
       LOOKUP=byname|bydns DATA=<text>
```

The data items can be given in any order, and all are optional. If no transport is included, the transport specified by the generic **transport** option is used. The host list and lookup type are needed only if the transport is an **smtp** transport that does not itself have a host list.

If the lookup type is not specified, Exim behaves as follows: First, a DNS lookup is done. If this yields anything other than HOST_NOT_FOUND, that result is used. Otherwise, Exim goes on to try a call to *getipnodebyname()* or *gethostbyname()*, and the result of the lookup is the result of that call.

If the DATA field is set, its value is placed in the **$address_data** variable. For example, this return line

```
accept hosts=x1.y.example:x2.y.example data="rule1"
```

routes the address to the default transport, with a host list containing two hosts. When the transport runs, the string ‘rule1’ is in **$address_data**.

21. The redirect router

The **redirect** router handles several kinds of address redirection. Its most common uses are for resolving local part aliases from a central alias file (usually called **/etc/aliases**) and for handling users' personal **.forward** files, but it has many other potential uses. The incoming address can be redirected in several different ways:

- It can be replaced by one or more new addresses which are themselves routed independently.
- It can be routed to be delivered to a given file or directory.
- It can be routed to be delivered to a specified pipe command.
- It can cause an automatic reply to be generated.
- It can be forced to fail, with a custom error message.
- It can be temporarily deferred.
- It can be discarded.

The generic **transport** option must not be set for **redirect** routers. However, there are some private options which define transports for delivery to files and pipes, and for generating autoreplies. See the **file_transport**, **pipe_transport** and **reply_transport** descriptions below.

21.1 Redirection data

The router operates by interpreting a text string which it obtains either by expanding the contents of the **data** option, or by reading the entire contents of a file whose name is given in the **file** option. These two options are mutually exclusive. The first is commonly used for handling system aliases, in a configuration like this:

```
system_aliases:
  driver = redirect
  data = ${lookup{$local_part}lsearch{/etc/aliases}}
```

If the lookup fails, causing the expanded string to be empty, the router declines. A configuration using **file** is commonly used for handling users' **.forward** files, like this:

```
userforward:
  driver = redirect
  check_local_user
  file = $home/.forward
  no_verify
```

If the file does not exist, or causes no action to be taken (for example, it is empty or consists only of comments), the router declines. **Warning**: This is not the case when the file contains syntactically valid items that happen to yield empty addresses, for example, items containing only RFC 2822 address comments.

21.2 Forward files and address verification

It is usual to set **no_verify** on **redirect** routers which handle users' **.forward** files, as in the example above. There are two reasons for this:

- When Exim is receiving an incoming SMTP message from a remote host, it is running under the Exim uid, not as root. Therefore, it is unable to change uid to read the file as the user, and it may not be able to read it as the Exim user. So in practice the router may not be able to operate.
- However, even when the router can operate, the existence of a **.forward** file is unimportant when verifying an address. What should be checked is whether the local part is a valid user name or not. Cutting out the redirection processing saves some resources.

21.3 Interpreting redirection data

The contents of the data string, whether obtained from **data** or **file**, can be interpreted in two different ways:

- If the **allow_filter** option is set true, and the data begins with the text '#Exim filter', it is interpreted as a list of *filtering* instructions in the form of an Exim filter file. Details of the syntax and semantics of filter files are described in a separate document entitled *Exim's interface to mail filtering*; this document is intended for use by end users.
- Otherwise, the data must be a comma-separated list of redirection items, as described in the next section.

21.4 Items in a non-filter redirection list

When the redirection data is not an Exim filter, for example, if it comes from a conventional alias or forward file, it consists of a list of addresses, file names, pipe commands, or certain special items (see below). The special items can be individually enabled or disabled by means of options whose names begin with **allow_** or **forbid_**, depending on their default values. The items in the list are separated by commas or newlines.

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 next newline character is ignored.

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.

Warning: If you use an Exim expansion to construct a redirection address, and the expansion contains a reference to **$local_part**, you should make use of the **quote** expansion operator, in case the local part contains special characters. For example, to redirect all mail for the domain *obsolete.example*, retaining the existing local part, you could use this setting:

```
data = ${quote:$local_part}@newdomain.example
```

21.5 Redirecting to a local mailbox

A redirection item may safely be the same as the address currently under consideration. This does not cause a routing loop, because a router is automatically skipped if any ancestor of the address that is being processed has the same local part and was processed by that router. The address is therefore passed to the following routers, so it is handled as if there were no redirection. When making this loop-avoidance test, the complete local part, including any prefix or suffix, is used.

Specifying the same local part without a domain is a common usage in personal filter files when the user wants to have messages delivered to the local mailbox and also forwarded elsewhere. For example, the user *spqr* might have a **.forward** file containing this:

```
spqr, Sam.Reman@other.domain.example
```

For compatibility with other MTAs, such local parts may be preceeded by '\', but this is not a requirement for loop prevention. However, it does make a difference if more than one domain is being handled synonymously.

If an item begins with '\' and the rest of the item parses as a valid RFC 2822 address that does not include a domain, the item is qualified using the domain of the incoming address. In the absence of a leading '\', unqualified addresses are qualified using the value in **qualify_recipient**, but you can force the incoming domain to be used by setting **qualify_preserve_domain**.

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.elsewhere.example
```

in *spqr*'s forward file fails on an incoming message addressed to *Sam.Reman*. The **redirect** router does not process *Sam.Reman* the second time round, because it has previously routed it, and the following routers presumably cannot handle the alias. The forward file should really contain

```
spqr, spqr@reme.elsewhere.example
```

but because this is such a common error, the **check_ancestor** option (see below) exists to provide a way to get round it. This is normally set on a **redirect** router that is handling users' **.forward** files.

21.6 Special items in redirection lists

In addition to addresses, the following types of item may appear in redirection lists:

- An item is treated as a pipe command if it begins with '|' and does not parse as a valid RFC 2822 address that includes a domain. A transport for running the command must be specified by the **pipe_transport** option. Either the router or the transport must specify a user and group under which to run the delivery. 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 in redirection lists 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 2822 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. Either the router or the transport must specify a user and group under which to run the delivery. However, if a redirection item is the path **/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.

- If an item is of the form

  ```
  :include:<path name>
  ```

  a list of further items is taken from the given file and included at that point. **Note**: such a file can not be a filter file; it is just an out-of-line addition to the list. The items in the included list are separated by commas or newlines and are not subject to expansion. If this is the first item in an alias list in an **lsearch** file, a colon must be used to terminate the alias name. This example is incorrect:

  ```
  list1    :include:/opt/lists/list1
  ```

  It must be given as

  ```
  list1:   :include:/opt/lists/list1
  ```

- Sometimes you want to throw away mail to a particular local part. Making the **data** option expand to an empty string does not work, because that causes the router to decline. Instead, the alias item

  ```
  :blackhole:
  ```

  can be used. It does what its name implies. No delivery is done, and no error message is generated. This has the same effect as specifing **/dev/null**, but can be independently disabled.

- An attempt to deliver a particular address can be deferred or forced to fail by redirection items of the form

  ```
  :defer:
  ```
  or
  ```
  :fail:
  ```

  respectively. When a redirection list contains such an item, it applies to the entire redirection; any other items in the list are ignored (*:blackhole:* is different). Any text following *:fail:* or *:defer:* is placed in the error text associated with the failure. For example, an alias file might contain:

  ```
  X.Employee:  :fail: Gone away, no forwarding address
  ```

  In the case of an address that is being verified from an ACL or as the subject of a VRFY command, the text is included in the SMTP error response by default. In an ACL, an explicitly provided message overrides the default, but the default message is available in the variable **$acl_verify_message** and can therefore be included in a custom message if this is desired. Exim sends a 451 SMTP code for a *:defer:*, and 550 for *:fail:*. In non-SMTP cases the text is included in the error message that Exim generates.

  Normally the error text is the rest of the redirection list – 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:* files.

  During routing for message delivery (as opposed to verification), a redirection 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 deferred for too long, it will ultimately fail, because the normal retry rules still apply.

- Sometimes it is useful to use a single-key search type with a default (see chapter 9) to look up 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 the **redirect** router to decline, whereas *:fail:* forces routing to fail. A lookup which results in an empty redirection list has the same effect.

21.7 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 routed to 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.

21.8 Repeated redirection expansion

When a message cannot be delivered to all of its recipients immediately, leading to two or more delivery attempts, redirection expansion is carried out afresh each time for those addresses whose children were not all previously delivered. If redirection 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.

21.9 Errors in redirection lists

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. Otherwise, if an error is detected while generating the list of new addresses, the original address is deferred. See also **syntax_errors_to**.

21.10 Private options for the redirect router

The private options for the **redirect** router are as follows:

allow_defer Type: *boolean* Default: *false*

Setting this option allows the use of *:defer:* in non-filter redirection data.

allow_fail Type: *boolean* Default: *false*

If this option is true, the *:fail:* item can be used in a redirection list, and the **fail** command may be used in a filter file.

allow_filter Type: *boolean* Default: *false*

Setting this option allows Exim to interpret redirection data that starts with '#Exim filter' as a set of filtering instructions. There are some features of filter files that some administrators may wish to lock out; see the **forbid_filter_xxx** options below. The filter is run using the uid and gid set by the generic **user** and **group** options. These take their defaults from the password data if **check_local_user** is set, so in the normal case of users' personal filter files, the filter is run as the relevant user. When **allow_filter** is set true, Exim insists that either **check_local_user** or **user** is set.

allow_freeze Type: *boolean* Default: *false*

Setting this option allows the use of the **freeze** command in a filter. This command is more normally encountered in system filters, and is disabled by default for redirection filters because it isn't something you usually want to let ordinary users do.

check_ancestor Type: *boolean* Default: *false*

This option is concerned with handling generated addresses which are the same as some address in the list of redirection ancestors of the current address. Although it is turned 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 **redirect** router.

When **check_ancestor** is set, if a generated address is the same as any ancestor of the current address, it is replaced by a copy of the current address. 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, <other item(s)>
```

Without the **check_ancestor** setting, either local part ('jb' or 'joe.bloggs') gets processed once by each router and so ends up as it was originally. If 'jb' is the real mailbox name, mail to 'jb' gets delivered (having been turned into 'joe.bloggs' by the **.forward** file and back to 'jb' by the alias), but mail to 'joe.bloggs' fails. Setting **check_ancestor** on the **redirect** router that handles the

.forward file prevents it from turning ‘jb’ back into ‘joe.bloggs’ when that was the original address. See also the **repeat_use** option below.

check_group Type: *boolean* Default: *see below*

When the **file** option is used, the group owner of the file is checked only when this option is set. The permitted groups are those listed in the **owngroups** option, together with the user’s default group if **check_local_user** is set. If the file has the wrong group, routing is deferred. The default setting for this option is true if **check_local_user** is set and the **modemask** option permits the group write bit, or if the **owngroups** option is set. Otherwise it is false, and no group check occurs.

check_owner Type: *boolean* Default: *see below*

When the **file** option is used, the owner of the file is checked only when this option is set. If **check_local_user** is set, the local user is permitted; otherwise the owner must be one of those listed in the **owners** option. The default value for this option is true if **check_local_user** or **owners** is set. Otherwise the default is false, and no owner check occurs.

data Type: *string*† Default: *unset*

This option is mutually exclusive with **file**. One or other of them must be set, but not both. The contents of **data** are expanded, and then used as the list of forwarding items, or as a set of filtering instructions. If the expansion is forced to fail, or the result is an empty string or a string that has no effect (consists entirely of comments), the router declines.

When filtering instructions are used, the string must begin with ‘#Exim filter’, and all comments in the string, including this initial one, must be terminated with newline characters. For example:

```
data = #Exim filter\n\
       if $h_to: contains Exim then save $home/mail/exim endif
```

If you are reading the data from a database where newlines cannot be included, you can use the **${sg}** expansion item to turn the escape string of your choice into a newline.

directory_transport Type: *string*† Default: *unset*

A **redirect** router 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. This should normally be an **appendfile** transport.

file Type: *string*† Default: *unset*

This option specifies the name of a file that contains the redirection data. It is mutually exclusive with the **data** option. The string is expanded before use; if the expansion is forced to fail, the router declines. Other expansion failures cause delivery to be deferred. The result of a successful expansion must be an absolute path. The entire file is read and used as the redirection data. If the data is an empty string or a string that has no effect (consists entirely of comments), the router declines.

If the attempt to open the file fails with a ‘does not exist’ error, Exim runs a check on the containing directory, unless **ignore_enotdir** is true (see below). If the directory does not appear to exist, delivery is deferred. This can happen when users’ **.forward** files are in NFS-mounted directories, and there is a mount problem. If the containing directory does exist, but the file does not, the router declines.

file_transport Type: *string*† Default: *unset*

A **redirect** router 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. This should normally be an **appendfile** transport. When it is running, the file name is in **$address_file**.

forbid_blackhole Type: *boolean* Default: *false*

If this option is true, the *:blackhole:* item may not appear in a redirection list.

forbid_file Type: *boolean* Default: *false*

If this option is true, this router may not generate a new address which specifies delivery to a local file or directory, either from a filter or from a conventional forward file. This option is forced to be true if **one_time** is set.

forbid_filter_existstest Type: *boolean* Default: *false*

If this option is true, string expansions in filters are not allowed to make use of the **exists** condition.

forbid_filter_logwrite Type: *boolean* Default: *false*

If this option is true, use of the logging facility in filters is not permitted. Logging 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).

forbid_filter_lookup Type: *boolean* Default: *false*

If this option is true, string expansions in filter files are not allowed to make use of **lookup** items.

forbid_filter_perl Type: *boolean* Default: *false*

This option is available only if Exim is built with embedded Perl support. If it is true, string expansions in filter files are not allowed to make use of the embedded Perl support.

forbid_filter_readfile Type: *boolean* Default: *false*

If this option is true, string expansions in filter files are not allowed to make use of **readfile** items.

forbid_filter_readsocket Type: *boolean* Default: *false*

If this option is true, string expansions in filter files are not allowed to make use of **readsocket** items.

forbid_filter_reply Type: *boolean* Default: *false*

If this option is true, this router may not generate an automatic reply message. Automatic replies can be generated only from filter files, not from traditional forward files. This option is forced to be true if **one_time** is set.

forbid_filter_run Type: *boolean* Default: *false*

If this option is true, string expansions in filter files are not allowed to make use of **run** items.

forbid_include Type: *boolean* Default: *false*

If this option is true, items of the form

```
:include:<path name>
```

are not permitted in non-filter redirection lists.

forbid_pipe Type: *boolean* Default: *false*

If this option is true, this router may not generate a new address which specifies delivery to a pipe, either from a filter or from a conventional forward file. This option is forced to be true if **one_time** is set.

hide_child_in_errmsg Type: *boolean* Default: *false*

If this option is true, it prevents Exim from quoting a child address if it generates a bounce or delay message for it. Instead it says 'an address generated from <*the top level address*>'. Of course, this applies only to bounces generated locally. If a message is forwarded to another host, *its* bounce may well quote the generated address.

ignore_eacces Type: *boolean* Default: *false*

If this option is set and an attempt to open a redirection file yields the EACCES error (permission denied), the **redirect** router behaves as if the file did not exist.

ignore_enotdir Type: *boolean* Default: *false*

If this option is set and an attempt to open a redirection file yields the ENOTDIR error (something on the path is not a directory), the **redirect** router behaves as if the file did not exist.

Setting **ignore_enotdir** has another effect as well: When a **redirect** router that has the **file** option set discovers that the file does not exist (the ENOENT error), it tries to *stat()* the parent directory, as a check against unmounted NFS directories. If the parent can not be statted, delivery is deferred. However, it seems wrong to do this check when **ignore_enotdir** is set, because that option tells Exim to ignore 'something on the path is not a directory' (the ENOTDIR error). This is a confusing area, because it seems that some operating systems give ENOENT where others give ENOTDIR.

include_directory Type: *string* Default: *unset*

If this option is set, the path names of any *:include:* items in a redirection list must start with this directory.

modemask Type: *octal integer* Default: *022*

This specifies mode bits which must not be set for a file specified by the **file** option. If any of the forbidden bits are set, delivery is deferred.

one_time Type: *boolean* Default: *false*

Sometimes the fact that Exim re-evaluates aliases and reprocesses redirection files each time it tries to deliver a message causes problems. This is particularly true in the case of mailing lists. If **one_time** is set and any addresses generated by the router 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, redirection does not happen again at the next delivery attempt.

Warning 1: This means that any header line addition or removal that is specified by this router would be lost if delivery did not succeed at the first attempt. For this reason, the **headers_add** and **headers_remove** generic options are not permitted when **one_time** is set.

Warning 2: To ensure that the router generates only addresses (as opposed to pipe or file deliveries or auto-replies) **forbid_file**, **forbid_pipe**, and **forbid_filter_reply** are forced to be true when **one_time** 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 **all_parents** log selector 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 specified by **file**. This list is in addition to the local user when **check_local_user** is set. See **check_owner** above.

owngroups Type: *string list* Default: *unset*

This specifies a list of permitted groups for the file specified by **file**. The list is in addition to the local user's primary group when **check_local_user** is set. See **check_group** above.

pipe_transport Type: *string*† Default: *unset*

A **redirect** router 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. This should normally be a **pipe** transport. When the transport is run, the pipe command is in **$address_pipe**.

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**.

repeat_use Type: *boolean* Default: *true*

If this option is set false, the router is skipped for a child address that has any ancestor that was routed by this router. This test happens before any of the other preconditions are tested. Exim's default anti-looping rules skip only when the ancestor is the same as the current address. See also **check_ancestor** above and the generic **redirect_router** option.

reply_transport Type: *string*† Default: *unset*

A **redirect** router sets up an automatic reply 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. This should normally be an **autoreply** transport. Other transports are unlikely to do anything sensible or useful.

rewrite Type: *boolean* Default: *true*

If this option is set false, addresses generated by the router are not subject to address rewriting. Otherwise, they are treated like new addresses and are rewritten according to the global rewriting rules.

skip_syntax_errors Type: *boolean* Default: *false*

If **skip_syntax_errors** is set, syntactically malformed addresses in non-filter redirection data are skipped, and each failing address is logged. If **syntax_errors_to** is set, a message is sent to the address it defines, giving details of the failures. If **syntax_errors_text** is set, its contents are expanded and placed at the head of the error message generated by **syntax_errors_to**. Usually it is appropriate to set **syntax_errors_to** to be the same address as the generic **errors_to** option. The **skip_syntax_errors** option is often used when handling mailing lists.

If all the addresses in a redirection list are skipped because of syntax errors, the router declines to handle the original address, and it is passed to the following routers.

If **skip_syntax_errors** is set when an Exim filter is interpreted, any syntax error in the filter causes filtering to be abandoned without any action being taken. The incident is logged, and the router declines to handle the address, so it is passed to the following routers.

skip_syntax_errors can be used to specify that errors in users' forward lists or filter files should not prevent delivery. The **syntax_errors_to** option, used with an address that does not get redirected, can be used to notify users of these errors, by means of a router like this:

```
userforward:
  driver = redirect
  allow_filter
  check_local_user
  file = $home/.forward
  file_transport = address_file
  pipe_transport = address_pipe
  reply_transport = address_reply
  no_verify
  skip_syntax_errors
  syntax_errors_to = real-$local_part@$domain
  syntax_errors_text = \
    This is an automatically generated message. An error has\n\
    been found in your .forward file. Details of the error are\n\
    reported below. While this error persists, you will receive\n\
    a copy of this message for every message that is addressed\n\
    to you. If your .forward file is a filter file, or if it is\n\
    a non-filter file containing no valid forwarding addresses,\n\
```

```
a copy of each incoming message will be put in your normal\n\
mailbox. If a non-filter file contains at least one valid\n\
forwarding address, forwarding to the valid addresses will\n\
happen, and those will be the only deliveries that occur.
```

You also need a router to ensure that local addresses that are prefixed by `real-` are recognized, but not forwarded or filtered. For example, you could put this immediately before the **userforward** router:

```
real_localuser:
  driver = accept
  check_local_user
  local_part_prefix = real-
  transport = local_delivery
```

syntax_errors_text Type: *string*† Default: *unset*

See **skip_syntax_errors** above.

syntax_errors_to Type: *string* Default: *unset*

See **skip_syntax_errors** above.

22. 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.) Exim always runs transports in subprocesses, under specified uids and gids. Typical deliveries to local mailboxes run under the uid and gid of the local user.

Exim also sets a specific current directory while running the transport; for some transports a home directory setting is also relevant. The **pipe** transport is the only one which sets up environment variables; see section 28.3 for details.

The values used for the uid, gid, and the directories may come from several different places. In many cases, the router that handles the address associates settings with that address as a result of its **check_local_user**, **group**, or **user** options. However, values may also be given in the transport's own configuration, and these override anything that comes from the router.

22.1 Uids and gids

All transports have the options **group** and **user**. If **group** is set, it overrides any group that the router set in the address, even if **user** is not set for the transport. This makes it possible, for example, to run local mail delivery under the uid of the recipient (set by the router), but in a special group (set by the transport). For example:

```
# Routers ...
# User/group are set by check_local_user in this router
local_users:
  driver = accept
  check_local_user
  transport = group_delivery

# Transports ...
# This transport overrides the group
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, **group** must be set.

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. When the uid is not specified by the transport, but is associated with the address by a router, the option for calling *initgroups()* is taken from the router configuration.

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.

22.2 Current and home directories

Routers may set current and home directories for local transports by means of the **transport_current_directory** and **transport_home_directory** options. However, if the transport's **current_directory** or **home_directory** options are set, they override the router's values. In detail, the home directory for a local transport is taken from the first of these values that is set:

- The **home_directory** option on the transport;
- The **transport_home_directory** option on the router;

- The password data if **check_local_user** is set on the router;
- The **router_home_directory** option on the router.

The current directory is taken from the first of these values that is set:

- The **current_directory** option on the transport;
- The **transport_current_directory** option on the router.

If neither the router nor the transport sets a current directory, Exim uses the value of the home directory, if it is set. Otherwise it sets the current directory to / before running a local transport.

22.3 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.

23. Generic options for transports

The following generic options apply to all transports:

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 **message_prefix** and **message_suffix** should be checked, because this option does not automatically suppress them.

current_directory Type: *string*† Default: *unset*

This specifies the current directory that is to be set while running the transport, overriding any value that may have been set by the router.

disable_logging Type: *boolean* Default: *false*

If this option is set true, nothing is logged for any transport errors. You should not set this option unless you really, really know what you are doing.

debug_print Type: *string*† Default: *unset*

If this option is set and debugging is enabled (see the **-d** command line option), 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. 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 the transport is configured to handle several addresses at once, or if more than one original address was redirected 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.

group Type: *string*† Default: *Exim group*

This option specifies a gid for running the transport process, overriding any value that the router supplies, and also overriding any value associated with **user** (see below).

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 2822 header lines, separated by newlines (coded as '\n'), 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. This is not recommended. Additional headers can also be specified by routers. See chapter 14 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 **message_prefix** and **message_suffix** should be checked, since this option does not automatically suppress them.

headers_remove Type: *string*† Default: *unset*

This option is expanded; the result must consist of a colon-separated list of header names, not including the terminating colon, for example:

```
headers_remove = return-receipt-to:acknowledge-to
```

Any existing headers matching those names are not included in any message that is transmitted by the transport. If there are multiple instances of a header, they are all removed. However, added headers may have these names. Thus it is possible to replace a header by specifying it in **headers_remove** and supplying the replacement in **headers_add**. Headers to be removed can also be specified by routers. See chapter 14 and section 43.13.

headers_rewrite Type: *string* Default: *unset*

This option allows addresses in header lines to be rewritten at transport time, that is, as the message is being copied to its destination. The contents of the option are a colon-separated list of rewriting rules. Each rule is in exactly the same form as one of the general rewriting rules that are applied when a message is received. These are described in chapter 30. For example,

```
headers_rewrite = a@b c@d f : \
                  x@y w@z
```

changes **a@b** into **c@d** in *From:* header lines, and **x@y** into **w@z** in all address-bearing header lines. The rules are applied to the header lines just before they are written out at transport time, so they affect only those copies of the message that pass through the transport. However, only the message's original header lines, and any that were added by a system filter, are rewritten. If a router or transport adds header lines, they are not affected by this option. These rewriting rules are *not* applied to the envelope. You can change the return path using **return_path**, but you cannot change envelope recipients at this time.

home_directory Type: *string*† Default: *unset*

This option specifies a home directory setting for the transport, overriding any value that may be set by the router. The home directory is placed in **$home** while expanding the transport's private options. It is also used as the current directory if no current directory is set by the **current_directory** option on the transport or the **transport_current_directory** option on the router.

initgroups Type: *boolean* Default: *false*

If this option is true and the uid for the delivery process is provided by the transport, the *initgroups()* function is called when running the transport to ensure that any additional groups associated with the uid are set up.

message_size_limit Type: *string*† Default: *0*

This option controls the size of messages passed through the transport. It is expanded before use; any expansion failure causes delivery to be deferred. The result of the expansion must be a sequence of digits, optionally followed by K or M. If its value is greater than zero and the size of a 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.

retry_use_local_part Type: *boolean* Default: *see below*

When a delivery suffers a temporary failure, a retry record is created in Exim's hints database. For remote deliveries, the key for the retry record is based on the name and/or IP address of the failing remote host. For local deliveries, the key is normally the entire address, including both the local part and the domain. This is suitable for most common cases of local delivery temporary failure – for example, exceeding a mailbox quota should delay only deliveries to that mailbox, not to the whole domain.

However, in some special cases you may want to treat a temporary local delivery as a failure associated with the domain, and not with a particular local part. (For example, if you are storing all mail for some domain in files.) You can do this by setting **retry_use_local_part** false.

For all the local transports, its default value is true. For remote transports, the default value is false for tidiness, but changing the value has no effect on a remote transport in the current implementation.

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 in the copy of the message that is being delivered. An empty return path is permitted. This feature is designed for remote deliveries, where the value of this option is used in the SMTP MAIL command. If you set **return_path** for a local transport, the only effect is to change the address that is placed in the *Return-path:* header line, if one is added to the message (see the next option).

The expansion can refer to the existing value via **$return_path**. This is either the message's envelope sender, or an address set by the **errors_address** option on a router. If the expansion is forced to fail, no replacement occurs; if it fails for another reason, delivery is deferred. This option can be used to support VERP (Variable Envelope Return Paths) – see chapter 42.

Note: If a delivery error is detected locally, the bounce message is not sent to the value of this option, but to the previously set errors address.

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 2821 states that the *Return-path:* header is added to a message 'when the delivery SMTP server makes the final delivery'. This 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*

A local transport may set the **shadow_transport** option to the name of another 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, with the same delivery address or addresses. 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 or via a system filter.

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 header lines, is passed to it on its standard input (this in fact is done from a third process, to avoid deadlock). The command must be specified as an absolute path.

The message is passed to the filter before any SMTP-specific processing, such as turning '\n' into '\r\n' and escaping lines beginning with a dot, and also before any processing implied by the settings of **check_string** and **escape_string** in the **appendfile** or **pipe** transports.

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 2822 syntax. A demonstration Perl script is provided in **util/transport-filter.pl**; this makes a few arbitrary modifications just to show the possibilities. Exim does not check the result, except to test for a final newline when SMTP is in use. All messages transmitted over SMTP must end with a newline, so Exim supplies one if it is missing.

A problem might arise if the filter increases the size of a message that is being sent down an SMTP connection. 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 contain the name and IP address of the host to which the message is being sent. 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 by default.

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.

user Type: *string*† Default: *Exim user*

This option specifies the user under whose uid the delivery process is to be run, overriding any uid that may have been set by the router. If the user is given as a name, the uid is looked up from the password data, and the associated group is taken as the value of the gid to be used if the **group** option is not set.

For remote transports, you should leave this option unset unless you really are sure you know what you are doing. When a remote transport is running, it needs to be able to access Exim's hints databases, because each host may have its own retry data.

24. Address batching in local transports

The only remote transport (**smtp**) is normally configured to handle more than one address at a time, so that when several addresses are routed to the same remote host, just one copy of the message is sent. Local transports, however, normally handle one address at a time. That is, a separate instance of the transport is run for each address that is routed to the transport. A separate copy of the message is delivered each time.

In special cases, it may be desirable to handle several addresses at once in a local transport, for example:

- In an **appendfile** transport, when storing messages in files for later delivery by some other means, a single copy of the message with multiple recipients saves space.
- In an **lmtp** transport, when delivering over 'local SMTP' to some process, a single copy saves time, and is the normal way LMTP is expected to work.
- In a **pipe** transport, when passing the message to a scanner program or to some other delivery mechanism such as UUCP, multiple recipients may be acceptable.

The three local transports (**appendfile**, **lmtp**, and **pipe**) all have the same options for controlling multiple ('batched') deliveries, namely **batch_max** and **batch_id**. To save repeating the information for each transport, these options are described here.

The **batch_max** option specifies the maximum number of addresses that can be delivered together in a single run of the transport. Its default value is one. When more than one address is routed to a transport that has a **batch_max** value greater than one, the addresses are delivered in a batch (that is, in a single run of the transport), subject to certain conditions:

- If any of the transport's options contain a reference to **$local_part**, no batching is possible.
- If any of the transport's options contain a reference to **$domain**, only addresses with the same domain are batched.
- If **batch_id** is set, it is expanded for each address, and only those addresses with the same expanded value are batched. This allows you to specify customized batching conditions.
- Batched addresses must also have the same errors address (where to send delivery errors), the same header additions and removals, the same user and group for the transport, and if a host list is present, the first host must be the same.

If the generic **envelope_to_add** option is set for the transport, the *Envelope-to:* header that is added to the message contains all the addresses that are batched together.

The **appendfile** and **pipe** transports have an option called **use_bsmtp**, which causes them to deliver the message in 'batched SMTP' format, with the envelope represented as SMTP commands. The **check_string** and **escape_string** options are forced to the values

```
check_string = "."
escape_string = ".."
```

when batched SMTP is in use. A full description of the batch SMTP mechanism is given in section 42.11. The **lmtp** transport does not have a **use_bsmtp** option, because it always delivers using the SMTP protocol.

If you are not using BSMTP, but are using a **pipe** transport, you can include **$pipe_addresses** as part of the command. This is not a true variable; it is a bit of magic that causes each of the recipient addresses to be inserted into the command as a separate argument. This provides a way of accessing all the addresses that are being delivered in the batch.

If you are using a batching **appendfile** transport without **use_bsmtp**, the only way to preserve the recipient addresses is to set the **envelope_to_add** option. This causes an *Envelope-to:* header line to be added to the message, containing all the recipients.

25. The appendfile transport

The **appendfile** transport delivers a message by appending it to an existing file, 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.

Exim recognises system quota errors, and generates an appropriate message. Exim also supports its own quota control within the transport, for use when the system facility is unavailable or cannot be used for some reason.

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. If there is an error while creating an entirely new file, the new file is removed.

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.

appendfile is most commonly used for local deliveries to users' mailboxes. However, it can also be used as a pseudo-remote transport for putting messages into files for remote delivery by some means other than Exim. 'Batch SMTP' format is often used in this case (see the **use_bsmtp** option).

25.1 The file and directory options

The **file** option specifies a single file, to which the message is appended; the **directory** option specifies a directory, in which a new file containing the message is created. Only one of these two options can be set, and for normal deliveries to mailboxes, one of them *must* be set.

However, **appendfile** is also used for delivering messages to files or directories whose names (or parts of names) are obtained from alias, forwarding, or filtering operations (for example, a **save** command in a user's Exim filter). When such a transport is running, **$local_part** contains the local part that was aliased or forwarded, and **$address_file** contains the name (or partial name) of the file or directory generated by the redirection operation. There are two cases:

- If neither **file** nor **directory** is set, the redirection operation must specify an absolute path (one that begins with `/`). This is the most common case when users with local accounts use filtering to sort mail into different folders. See for example, the **address_file** transport in the default configuration. If the path ends with a slash, it is assumed to be the name of a directory. A delivery to a directory can also be forced by setting **maildir_format** or **mailstore_format**.
- If **file** or **directory** is set for a delivery from a redirection, it is used to determine the file or directory name for the delivery. Normally, the contents of **$address_file** are used in some way in the string expansion. For example, in an environment where users do not have home directories, they may be permitted to use filter commands of the form:

  ```
  save inbox.folder23
  ```

  The expansion of **file** in the transport could transform **inbox.folder23** into the appropriate file name.

Note 1: A relative path such as **inbox.folder23** is turned into an absolute path by the **redirect** router if a home directory exists. If you want to prevent this, you can set **router_home_directory** empty.

Note 2: An absolute path in **$address_file** is not treated specially; the **file** or **directory** option is still used if it is set.

25.2 Private options for appendfile

allow_fifo Type: *boolean* Default: *false*

Setting this option permits delivery to named pipes (FIFOs) as well as to regular files. If no process is reading the named pipe at delivery time, the delivery is deferred.

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_id Type: *string†* Default: *unset*

See the description of local delivery batching in chapter 24. However, batching is automatically disabled for **appendfile** deliveries that happen as a result of forwarding or aliasing or other redirection directly to a file.

batch_max Type: *integer* Default: *1*

See the description of local delivery batching in chapter 24.

check_group Type: *boolean* Default: *false*

When this option is set, the group owner of the file defined by the **file** option is checked to see that it is the same as the group under which the delivery process is running. The default setting is false because the default file mode is 0600, which means that the group is irrelevant.

check_owner Type: *boolean* Default: *true*

When this option is set, the owner of the file defined by the **file** option is checked to ensure that it is the same as the user under which the delivery process is running.

check_string Type: *string* Default: *see below*

As **appendfile** writes the message, the start of each line is tested for matching **check_string**, and if it does, the initial matching characters are replaced by the contents of **escape_string**. The value of **check_string** is a literal string, not a regular expression, and the case of any letters it contains is significant.

If **use_bsmtp** is set the values of **check_string** and **escape_string** are forced to '.' and '..' respectively, and any settings in the configuration are ignored. Otherwise, they default to 'From ' and '>From ' when the **file** option is set, and unset when any of the **directory**, **maildir**, or **mailstore** options are set.

The default settings, along with **message_prefix** and **message_suffix**, are suitable for traditional 'BSD' mailboxes, where a line beginning with 'From ' indicates the start of a new message. All four options need changing if another format is used. For example, to deliver to mailboxes in MMDF format:

```
check_string = "\1\1\1\1\n"
escape_string = "\1\1\1\1 \n"
message_prefix = "\1\1\1\1\n"
message_suffix = "\1\1\1\1\n"
```

create_directory Type: *boolean* Default: *true*

When this option is true, Exim attempts to create 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 and directories that are created by this transport. It applies to files defined by the **file** option and directories defined by the **directory** option. In the case of maildir delivery, it applies to the top level directory, not the maildir directories beneath.

The option 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 for the transport. This option is not useful when an explicit file name is given for normal mailbox deliveries. It is intended for the case when file names are generated from users' **.forward** files. These are usually handled by an **appendfile** transport called **address_file**. See also **file_must_exist**.

directory Type: *string*† Default: *unset*

This option is mutually exclusive with the **file** option, but one of **file** or **directory** must be set, unless the delivery is the direct result of a redirection (see section 25.1).

When **directory** 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. A number of different formats are provided (see **maildir_format** and **mailstore_format**), and see section 25.4 for further details of this form of delivery.

directory_file Type: *string*†
Default: `q${base62:$tod_epoch}-$inode`

When **directory** is set, but neither **maildir_format** nor **mailstore_format** is set, **appendfile** delivers each message into a file whose name is obtained by expanding this string. The default value generates a unique name from the current time, in base 62 form, and the inode of the file. The variable **$inode** is available only when expanding this option.

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.

escape_string Type: *string* Default: *see description*

See **check_string** above.

file Type: *string*† Default: *unset*

This option is mutually exclusive with the **directory** option, but one of **file** or **directory** must be set, unless the delivery is the direct result of a redirection (see section 25.1). The **file** option specifies a single file, to which the message is appended. One or more of **use_fcntl_lock**, **use_flock_lock**, or **use_lockfile** 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. 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.

file_format Type: *string* Default: *unset*

This option requests the transport to check the format of an existing file before adding to it. The check consists of matching a specific string at the start of the file. The value of the option consists of an even number of colon-separated strings. The first of each pair is the test string, and the second is the name of a transport. If the transport associated with a matched string is not the current transport, control is passed over to the other transport. For example, suppose the standard **local_delivery** transport has this added to it:

```
file_format = "From       : local_delivery :\
               \1\1\1\1\n : local_mmdf_delivery"
```

Mailboxes that begin with 'From' are still handled by this transport, but if a mailbox begins with four binary ones followed by a newline, control is passed to a transport called **local_mmdf_delivery**, which presumably is configured to do the delivery in MMDF format. If a mailbox does not exist or is empty, it is assumed to match the current transport. If the start of a mailbox doesn't match any string, or if the transport named for a given string is not defined, delivery is deferred.

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.

lock_fcntl_timeout Type: *time* Default: *0s*

By default, the **appendfile** transport uses non-blocking calls to *fcntl()* when locking an open mailbox file. If the call fails, the delivery process sleeps for **lock_interval** and tries again, up to **lock_retries** times. Non-blocking calls are used so that the file is not kept open during the wait for the lock; the reason for this is to make it as safe as possible for deliveries over NFS in the case when processes might be accessing an NFS mailbox without using a lock file. This should not be done, but misunderstandings and hence misconfigurations are not unknown.

On a busy system, however, the performance of a non-blocking lock approach is not as good as using a blocking lock with a timeout. In this case, the waiting is done inside the system call, and Exim's delivery process acquires the lock and can proceed as soon as the previous lock holder releases it.

If **lock_fcntl_timeout** is set to a non-zero time, blocking locks, with that timeout, are used. There may still be some retrying: the maximum number of retries is

```
(lock_retries * lock_interval) / lock_fcntl_timeout
```

rounded up to the next whole number. In other words, the total time during which **appendfile** is trying to get a lock is roughly the same, unless **lock_fcntl_timeout** is set very large.

You should consider setting this option if you are getting a lot of delayed local deliveries because of errors of the form

```
failed to lock mailbox /some/file (fcntl)
```

lock_flock_timeout Type: *time* Default: *0s*

This timeout applies to file locking when using *flock()* (see **use_flock**); the timeout operates in a similar manner to **lock_fcntl_timeout**.

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, the delivery is into a new file, in the ‘maildir’ format that is used by other mail software. When the transport is activated directly from a **redirect** router (for example, the **address_file** transport in the default configuration), setting **maildir_format** causes the path received from the router to be treated as a directory, whether or not it ends with /. This option is available only if SUPPORT_MAILDIR is present in **Local/Makefile**. See section 25.4 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 25.4 below.

maildir_tag Type: *string*† Default: *unset*

This option applies only to deliveries in maildir format, and is described in section 25.4 below.

mailstore_format Type: *boolean* Default: *false*

If this option is set with the **directory** option, 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 25.4 below for further details.

mailstore_prefix Type: *string*† Default: *unset*

This option applies only to deliveries in mailstore format, and is described in section 25.4 below.

mailstore_suffix Type: *string*† Default: *unset*

This option applies only to deliveries in mailstore format, and is described in section 25.4 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, 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, by means of the *c-client* library that they all use. The **message_prefix** and **message_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.

message_prefix Type: *string*† Default: *see below*

The string specified here is expanded and output at the start of every message. The default is unset unless **file** is specified and **use_bsmtp** is not set, in which case it is:

```
message_prefix = "From ${if def:return_path{$return_path}\
  {MAILER-DAEMON}} $tod_bsdinbox\n"
```

message_suffix Type: *string*† Default: *see below*

The string specified here is expanded and output at the end of every message. The default is unset unless **file** is specified and **use_bsmtp** is not set, in which case it is a single newline character. The suffix can be suppressed by setting

```
message_suffix =
```

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, 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 deferred (‘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.

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 when the **directory** option is set. In the latter case, computation of the space used is expensive, because all the files in the directory (and any sub-directories) have to be individually inspected and their sizes summed (but see **quota_size_regex** below). Also, there is no interlock against two simultaneous deliveries into a multi-file mailbox. For single-file mailboxes, of course, an interlock is a necessity.

A file’s size is taken as its *used* value. Because of blocking effects, this may be a lot less than the actual amount of disk space allocated to the file. If the sizes of a number of files are being added up, the rounding effect can become quite noticeable, especially on systems that have large block sizes. Nevertheless, it seems best to stick to the *used* figure, because this is the obvious value which users understand most easily.

The value of the option is expanded, and must then be a numerical value (decimal point allowed), optionally followed by one of the letters K or M. A value of zero unsets the option. The expansion happens while Exim is running as root, before it changes uid for the delivery. This means that files which 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.

By default, Exim’s quota checking mimics system quotas, and restricts the mailbox to the specified maximum size, though the value is not accurate to the last byte, owing to separator lines and additional headers that may get added during message delivery. When a mailbox is nearly full, large messages may get refused even though small ones are accepted, because the size of the current message is added to the quota when the check is made. This behaviour can be changed by setting **quota_is_inclusive** false. When this is done, the check for exceeding the quota does not include the

current message. Thus, deliveries continue until the quota has been exceeded; thereafter, no further messages are delivered. See also **quota_warn_threshold**.

quota_directory Type: *string†* Default: *unset*

This option defines the directory to check for quota purposes when delivering into individual files. The default is the delivery directory, or, if a file called **maildirfolder** exists in a maildir directory, the parent of the delivery directory.

quota_filecount Type: *string†* 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. The value is expanded; an expansion failure causes delivery to be deferred.

quota_is_inclusive Type: *boolean* Default: *true*

See **quota** above.

quota_size_regex Type: *string* Default: *unset*

This option applies when one of the delivery modes that writes a separate file for each message is being used. When Exim wants to find the size of one of these files in order to test the quota, it first checks **quota_size_regex**. If this is set to a regular expression that matches the file name, and it captures one string, that string is interpreted as a representation of the file's size. The value of **quota_size_regex** is not expanded.

This feature is useful only when users have no shell access to their mailboxes – otherwise they could defeat the quota simply by renaming the files. This facility can be used with maildir deliveries, by setting **maildir_tag** to add the file length to the file name. For example:

```
maildir_tag = ,S=$message_size
quota_size_regex = ,S=(\d+)
```

The regular expression should not assume that the length is at the end of the file name (even though **maildir_tag** puts it there) because maildir MUAs sometimes add other information onto the ends of message file names.

quota_warn_message Type: *string†* Default: *see below*

See below 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"
```

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, a warning message is sent. If **quota** is also set, the threshold may be specified as a percentage of it by following the value with a percent sign. For example:

```
quota = 10M
quota_warn_threshold = 75%
```

If **quota** is not set, a setting of **quota_warn_threshold** that ends with a percent sign is ignored.

The warning 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 except when the threshold is specified as a percentage.

use_bsmtp Type: *boolean* Default: *false*

If this option is set true, **appendfile** writes messages in 'batch SMTP' format, with the envelope sender and recipient(s) included as SMTP commands. If you want to include a leading HELO command with such messages, you can do so by setting the **message_prefix** option. See section 42.11 for details of batch SMTP.

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 **message_prefix** and **message_suffix** options are written verbatim, so must contain their own carriage return characters if these are needed. In cases where these options have non-empty defaults, the values end with a single linefeed, so they almost always need to be changed to end with `\r\n` 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_flock_lock** is set. Otherwise, it should be turned off only if you know that all your MUAs use lock file locking. When both **use_fcntl_lock** and **use_flock_lock** are unset, **use_lockfile** must be set.

use_flock_lock Type: *boolean* Default: *false*

This option is provided to support the use of *flock()* for file locking, for the few situations where it is needed. Most modern operating systems support *fcntl()* and *lockf()* locking, and these two functions interwork with each other. Exim uses *fcntl()* locking by default.

This option is required only if you are using an operating system where *flock()* is used by programs that access mailboxes (typically MUAs), and where *flock()* does not correctly interwork with *fcntl()*. You can use both *fcntl()* and *flock()* locking simultaneously if you want.

Not all operating systems provide *flock()*. Some versions of Solaris do not have it (and some, I think, provide a not quite right version built on top of *lockf()*). If the OS does not have *flock()*, Exim will be built without the ability to use it, and any attempt to do so will cause a configuration error.

Warning: *flock()* locks do not work on NFS files (unless *flock()* is just being mapped onto *fcntl()* by the OS).

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 mailbox file. In this situation, the only locking is by *fcntl()*. 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.

The **use_lockfile** 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.

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 Pine 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.

You can set **use_mbx_lock** with either (or both) of **use_fcntl_lock** and **use_flock_lock** to control what kind of locking is used in implementing the MBX locking rules. The default is to use *fcntl()* if **use_mbx_lock** is set without **use_fcntl_lock** or **use_flock_lock**.

25.3 Operational details for appending

Before appending to a file, the following preparations are made:

- 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 path 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 **file_format** is set, the format of an existing file is checked. If this indicates that a different transport should be used, control is passed to that transport.
- 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, 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, 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.
- If the file's permissions are more generous than specified, they are reduced. If they are insufficient, delivery is deferred, unless **mode_fail_narrower** is set false, in which case the delivery is tried using the existing permissions.

- The file's inode number is saved, and the file is then opened for appending. If this fails because the file has vanished, **appendfile** behaves as if it hadn't existed (see below). For any other failures, delivery is deferred.
- 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 delivery and freeze the message.
- If the file did not exist originally, defer delivery 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 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_flock_lock** are false, it is locked using *fcntl()* or *flock()* or both. If **use_mbx_lock** is false, an exclusive lock is requested in each case. However, if **use_mbx_lock** is true, 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 Exim fails to lock the file, there are two possible courses of action, depending on the value of the locking timeout. This is obtained from **lock_fcntl_timeout** or **lock_flock_timeout**, as appropriate.

  If the timeout value is zero, the file is closed, Exim waits for **lock_interval**, and then goes back and re-opens the file as above and tries to lock it again. This happens up to **lock_retries** times, after which the delivery is deferred.

  If the timeout has a value greater than zero, blocking calls to *fcntl()* or *flock()* are used (with the given timeout), so there has already been some waiting involved by the time locking fails. Nevertheless, Exim does not give up immediately. It retries up to

  `(lock_retries * lock_interval) / `<*timeout*>

  times (rounded up).

At the end of delivery, Exim closes the file (which releases the *fcntl()* and/or *flock()* locks) and then deletes the lock file if one was created.

25.4 Operational details for delivery to a new file

When the **directory** option is set instead of **file**, each message is delivered into a newly-created file or set of files. When **appendfile** is activated directly from a **redirect** router, neither **file** nor **directory** is normally set, because the path for delivery is supplied by the router. (See for example, the **address_file** transport in the default configuration.) In this case, delivery is to a new file if either the path name ends in `/`, or the **maildir_format** or **mailstore_format** option is set.

No locking is required while writing the message to a new file, 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 values for **check_string**, **message_prefix**, and **message_suffix** are all unset when any of **directory**, **maildir_format**, or **mailstore_format** is set.

If Exim is required to check a **quota** setting, it adds up the sizes of all the files in the delivery directory by default. However, you can specify a different directory by setting **quota_directory**. Also, for maildir deliveries (see below) the **maildirfolder** convention is honoured.

There are three different ways in which delivery to individual files can be done, controlled by the settings of the **maildir_format** and **mailstore_format** options. Note that code to support maildir or mailstore formats is not included in the binary unless SUPPORT_MAILDIR or SUPPORT_MAILSTORE, respectively, is 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). The location of a created directory can be constrained by setting **create_file**. 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, delivery is deferred.

25.5 Maildir delivery

If the **maildir_format** option is true, Exim delivers each message by writing it to a file whose name is **tmp/**<*stime*>**.H**<*mtime*>**P**<*pid*>**.**<*host*> in the given directory. If the delivery is successful, the file is renamed into the **new** subdirectory.

In the file name, <*stime*> is the current time of day in seconds, and <*mtime*> is the microsecond fraction of the time. After a maildir delivery, Exim checks that the time-of-day clock has moved on by at least one microsecond before terminating the delivery process. This guarantees uniqueness for the file name. However, as a precaution, Exim calls *stat()* for the file before opening it. If any response other than ENOENT (does not exist) is given, Exim waits 2 seconds and tries again, up to **maildir_retries** times.

If Exim is required to check a **quota** setting before a maildir delivery, and **quota_directory** is not set, it looks for a file called **maildirfolder** in the maildir directory (alongside **new**, **cur**, **tmp**). If this exists, Exim assumes the directory is a maildir++ folder directory, which is one level down from the user's top level mailbox directory. This causes it to start at the parent directory instead of the current directory when calculating the amount of space used.

If **maildir_tag** is set, the string is expanded for each delivery. When the maildir file is renamed into the **new** sub-directory, the tag is added to its name. However, if adding the tag takes the length of the name to the point where the test *stat()* call fails with ENAMETOOLONG, the tag is dropped and the maildir file is created with no tag.

Tags can be used to encode the size of files in their names; see **quota_size_regex** above for an example. The expansion of **maildir_tag** happens after the message has been written. The value of the **$message_size** variable is set to the number of bytes actually written. If the expansion is forced to fail, the tag is ignored, but a non-forced failure causes delivery to be deferred. The expanded tag may contain any printing characters except '/'. Non-printing characters in the string are ignored; if the resulting string is empty, it is ignored. If it starts with an alphanumeric character, a leading colon is inserted.

25.6 Mailstore delivery

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_max** option is set greater than one. 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 **mailstore_prefix** or **mailstore_suffix** ends with a forced failure, it is ignored. Other expansion errors are treated as serious configuration errors, and delivery is deferred.

25.7 Non-special new file delivery

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 in batched SMTP format for later delivery to some host (see section 42.11), 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 obtained by expanding the contents of the **directory_file** option.

26. 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. It is usually run as the result of mail filtering, a 'vacation' message being the standard example. However, it can also be run directly from a router like any other transport. To reduce the possibility of message cascades, messages created by the **autoreply** transport always have empty envelope sender addresses, like bounce messages.

The parameters of the message to be sent can be specified in the configuration by options described below. However, these are used only when the address passed to the transport does not contain its own reply information. When the transport is run as a consequence of a **mail** or **vacation** command in a filter file, the parameters of the message are supplied by the filter, and passed with the address. The transport's options that define the message are then ignored (so they are not usually set in this case). The message is specified entirely by the filter or by the transport; it is never built from a mixture of options. However, the **file_optional**, **mode**, and **return_message** options apply in all cases.

Autoreply is implemented as a local transport. When used as a result of a command in a user's filter file, **autoreply** normally runs under the uid and gid of the user, and with appropriate current and home directories (see chapter 22).

There is a subtle difference between routing a message to a **pipe** transport that generates some text to be returned to the sender, and routing 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, whereas if **autoreply** is used, a separate message is generated for each address that is passed to it.

Non-printing characters are not permitted in the header lines generated for the message that **autoreply** creates, with the exception of newlines that are immediately followed by whitespace. If any non-printing characters are found, the transport defers. Whether characters with the top bit set count as printing characters or not is controlled by the **print_topbitchars** global option.

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 bounce message) do not cause problems. They are just discarded.

26.1 Private options for autoreply

bcc Type: *string*† Default: *unset*

This specifies the addresses that are to receive 'blind carbon copies' of the message when the message is specified by the transport.

cc Type: *string*† Default: *unset*

This specifies recipients of the message and the contents of the *Cc:* header when the message is specified by the transport.

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. 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 or passed with the address does not exist or cannot be read.

from Type: *string†* Default: *unset*

This specifies the contents of the *From:* header when the message is specified by the transport.

headers Type: *string†* Default: *unset*

This specifies additional RFC 2822 headers that are to be added to the message when the message is specified by the transport. Several can be given by using '\n' to separate them. There is no check on the format.

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.

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 file or DBM database in which a record of each *To:* recipient is kept when the message is specified by the transport. **Note**: This does not apply to *Cc:* or *Bcc:* recipients. If **once_file_size** is not set, a DBM database is used, and it is allowed to grow as large as necessary. 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.

If **once_file_size** is set greater than zero, it changes the way Exim implements the **once** option. Instead of using a DBM file to record every recipient it sends to, it uses a regular file, whose size will never get larger than the given value. In the file, it keeps a linear list of recipient addresses and times at which they were sent messages. If the file is full when a new address needs to be added, the oldest address is dropped. If **once_repeat** is not set, this means that a given recipient may receive multiple messages, but at unpredictable intervals that depend on the rate of turnover of addresses in the file. If **once_repeat** is set, it specifies a maximum time between repeats.

once_file_size Type: *integer* Default: *0*

See **once** above.

once_repeat Type: *time†* Default: *0s*

See **once** above. After expansion, the value of this option must be a valid time value.

reply_to Type: *string†* Default: *unset*

This specifies the contents of the *Reply-To:* header when the message is specified by the transport.

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** global configuration option.

subject Type: *string*† Default: *unset*

This specifies the contents of the *Subject:* header when the message is specified by the transport.

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. If both **text** and **file** are set, the text comes first.

to Type: *string*† Default: *unset*

This specifies recipients of the message and the contents of the *To:* header when the message is specified by the transport.

27. The lmtp transport

The **lmtp** transport runs the LMTP protocol (RFC 2033) over a pipe to a specified command or by interacting with a Unix domain socket. This transport is something of a cross between the **pipe** and **smtp** transports. Exim also has support for using LMTP over TCP/IP; this is implemented as an option for the **smtp** transport. Because LMTP is expected to be of minority interest, the default build-time configure in **src/EDITME** has it commented out. You need to ensure that

```
TRANSPORT_LMTP=yes
```

is present in your **Local/Makefile** in order to have the **lmtp** transport included in the Exim binary.

The private options of the **lmtp** transport are as follows:

batch_id Type: *string*† Default: *unset*

See the description of local delivery batching in chapter 24.

batch_max Type: *integer* Default: *1*

This limits the number of addresses that can be handled in a single delivery. Most LMTP servers can handle several addresses at once, so it is normally a good idea to increase this value. See the description of local delivery batching in chapter 24.

command Type: *string*† Default: *unset*

This option must be set if **socket** is not set. The string is a command which is run in a separate process. It is split up into a command name and list of arguments, each of which is separately expanded (so expansion cannot change the number of arguments). The command is run directly, not via a shell. The message is passed to the new process using the standard input and output to operate the LMTP protocol.

socket Type: *string*† Default: *unset*

This option must be set if **command** is not set. The result of expansion must be the name of a Unix domain socket. The transport connects to the socket and delivers the message to it using the LMTP protocol.

timeout Type: *time* Default: *5m*

The transport is aborted if the created process or Unix domain socket does not respond to LMTP commands or message input within this timeout.

Here is an example of a typical LMTP transport:

```
lmtp:
  driver = lmtp
  command = /some/local/lmtp/delivery/program
  batch_max = 20
  user = exim
```

This delivers up to 20 addresses at a time, in a mixture of domains if necessary, running as the user *exim*.

28. The pipe transport

The **pipe** transport is used to deliver messages via a pipe to a command running in another process. This can happen in one of two ways:

- A router routes an address to a transport in the normal way, and the transport is configured as a **pipe** transport. In this case, **$local_part** contains the address (as usual), and the command which is run is specified by the **command** option on the transport. An example of this is the use of **pipe** as a pseudo-remote transport for passing messages to some other delivery mechanism (such as UUCP).
- A router redirects an address directly to a pipe command (for example, from an alias or forward file). In this case, **$local_part** contains the local part that was redirected, and **$address_pipe** contains the text of the pipe command itself. The **command** option on the transport is ignored.

The **pipe** transport is a non-interactive delivery method. Exim can also deliver messages over pipes using the LMTP interactive protocol. This is implemented by the **lmtp** transport.

In the case when **pipe** is run as a consequence of an entry in a local user's **.forward** file, the command runs under the uid and gid of that user. In other cases, the uid and gid have to be specified explicitly, either on the transport or on the router that handles the address. Current and 'home' directories are also controllable. See chapter 22 for details of the local delivery environment.

28.1 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.

Details of a permanent failure are logged, but are not included in the bounce message, which merely contains 'local delivery failed'.

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 streams, the command 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 included as part of the bounce message. The **return_fail_output** option is similar, except that 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**.

28.2 How the command is run

The command line is (by default) broken down into a command name and arguments by the **pipe** transport itself. 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. If an argument appears in double quotes, backslash is interpreted as an escape character in the usual way. If an argument appears in single quotes, no escaping is done.

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. For this reason, any string expansion item that contains white space must be quoted so as to be contained within a single argument. A setting such as

```
command = /some/path ${if eq{$local_part}{postmaster}{xxx}{yyy}}
```

will not work, because the expansion item gets split between several arguments. You have to write

```
command = /some/path "${if eq{$local_part}{postmaster}{xxx}{yyy}}"
```

to ensure that it is all in one argument. The expansion is done in this way, argument by argument, so that the number of arguments cannot be changed as a result of expansion, and quotes or backslashes in inserted variables do not interact with external quoting.

Special handling takes place when an argument consists of precisely 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.

After splitting up into arguments and expansion, the resulting command is run in a subprocess directly from the transport, *not* under a shell. The message that is being delivered is supplied on the standard input, and the standard output and standard error are 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.

28.3 Environment variables

The environment variables listed below are set up when the command is invoked. This list is a compromise for maximum compatibility with other MTAs. Note that the **environment** option can be used to add additional variables to this environment.

`DOMAIN`	the domain of the address
`HOME`	the home directory, if set
`HOST`	the host name when called from a router (see below)
`LOCAL_PART`	see below
`LOCAL_PART_PREFIX`	see below
`LOCAL_PART_SUFFIX`	see below
`LOGNAME`	see below
`MESSAGE_ID`	the message's id
`PATH`	as specified by the **path** option below
`QUALIFY_DOMAIN`	the sender qualification domain
`RECIPIENT`	the complete recipient address
`SENDER`	the sender of the message (empty if a bounce)
`SHELL`	'`/bin/sh`'
`TZ`	the value of the **timezone** option, if set
`USER`	see below

When a **pipe** transport is called directly from (for example) an **accept** router, 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. In both cases, any affixes are removed from the local part, and made available in LOCAL_PART_PREFIX and LOCAL_PART_SUFFIX, respectively. 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 that associates hosts with an address, typically when using **pipe** as a pseudo-remote transport. HOST is set to the first host name specified by the router.

If the transport's generic **home_directory** option is set, its value is used for the HOME environment variable. Otherwise, a home directory may be set by the router's **transport_home_directory** option, which defaults to the user's home directory if **check_local_user** is set.

28.4 Private options for pipe

allow_commands Type: *string list*† Default: *unset*

The string is expanded, and is then interpreted as a colon-separated list of permitted commands. If **restrict_to_path** is not set, 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**, 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/bin/vacation
```

and **restrict_to_path** is not set, the only permitted command is **/usr/bin/vacation**. The **allow_commands** option may not be set if **use_shell** is set.

batch_id Type: *string*† Default: *unset*

See the description of local delivery batching in chapter 24.

batch_max Type: *integer* Default: *1*

This limits the number of addresses that can be handled in a single delivery. See the description of local delivery batching in chapter 24.

check_string Type: *string* Default: *unset*

As **pipe** writes the message, the start of each line is tested for matching **check_string**, and if it does, the initial matching characters are replaced by the contents of **escape_string**, provided both are set. The value of **check_string** is a literal string, not a regular expression, and the case of any letters it contains is significant. When **use_bsmtp** is set, the contents of **check_string** and **escape_string** are forced to values that implement the SMTP escaping protocol. Any settings made in the configuration file are ignored.

command Type: *string*† Default: *unset*

This option need not be set when **pipe** is being used to deliver to pipes obtained directly from address redirections. 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, as described in section 28.2 above.

environment Type: *string*† Default: *unset*

This option is used to add additional variables to the environment in which the command runs (see section 28.3 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>*'.

escape_string Type: *string* Default: *unset*

See **check_string** above.

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**.

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 one of those listed in **temp_errors**, which causes the delivery to be deferred and tried again later.

log_defer_output Type: *boolean* Default: *false*

If this option is set, and the status returned by the command is one of the codes listed in **temp_errors** (that is, delivery was deferred), 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 one of the return codes listed in **temp_errors** (that is, the delivery failed), 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 independently of the settings of the options that control what is done with such output (for example, **return_output**). Because of buffering effects, the amount of output may exceed the limit by a small amount before Exim notices.

message_prefix Type: *string*† Default: *see below*

The string specified here is expanded and output at the start of every message. The default is unset if **use_bsmtp** is set. Otherwise it is

```
message_prefix = \
  From ${if def:return_path{$return_path}{MAILER-DAEMON}}\
  ${tod_bsdinbox}\n
```

This is required by the commonly used **/usr/bin/vacation** program. However, 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

```
message_prefix =
```

message_suffix Type: *string*† Default: *see below*

The string specified here is expanded and output at the end of every message. The default is unset if **use_bsmtp** is set. Otherwise it is a single newline. The suffix can be suppressed by setting

```
message_suffix =
```

path Type: *string* 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. **Warning**: This does not apply to a command specified as a transport filter.

pipe_as_creator Type: *boolean* Default: *false*

If the generic **user** option is not set and this option is true, 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 generic **group** option), the gid that was in force when Exim was originally called to accept the message is used.

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**.

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 one of the codes listed in **temp_errors** (that is, the delivery failed), the output is returned in the bounce message. However, if the message has a null sender (that is, it is itself a bounce 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 bounce message. Otherwise, the output is just discarded. However, if the message has a null sender (that is, it is a bounce message), output from the command is always discarded, whatever the setting of this option.

temp_errors Type: *string list* Default: *see below*

This option contains either a colon-separated list of numbers, or a single asterisk. If **ignore_status** is false and the command exits with a non-zero return code, the failure is treated as temporary and the delivery is deferred if the return code matches one of the numbers, or if the setting is a single asterisk. Otherwise, non-zero return codes are treated as permanent errors. 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_bsmtp Type: *boolean* Default: *false*

If this option is set true, the **pipe** transport writes messages in 'batch SMTP' format, with the envelope sender and recipient(s) included as SMTP commands. If you want to include a leading HELO command with such messages, you can do so by setting the **message_prefix** option. See section 42.11 for details of batch SMTP.

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 **message_prefix** and **message_suffix** options are written verbatim, so must contain their own carriage return characters if these are needed. Since the default values for both

message_prefix and **message_suffix** end with a single linefeed, their values almost always need to be changed to end with `\r\n` 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 28.2. 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.

28.5 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 router configuration for **procmail**:

```
# transport
procmail_pipe:
  driver = pipe
  command = /usr/local/bin/procmail -d $local_part
  return_path_add
  delivery_date_add
  envelope_to_add
  check_string = "From "
  escape_string = ">From "
  user = $local_part
  group = mail

# router
procmail:
  driver = accept
  check_local_user
  transport = procmail_pipe
```

In this example, the pipe is run as the local user, but with the group set to *mail*. An alternative is to run the pipe as a specific user such as *mail* or *exim*, but in this case you must arrange for **procmail** to trust that user to supply a correct sender address. If you do not specify either a **group** or a **user** option, 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 router 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
  message_prefix =
  message_suffix =

# router
local_user_cyrus:
  driver = accept
  check_local_user
  local_part_suffix = .*
  transport = local_delivery_cyrus
```

Note the unsetting of **message_prefix** and **message_suffix**, and the use of **return_output** to cause any text written by Cyrus to be returned to the sender.

29. The smtp transport

The **smtp** transport delivers messages over TCP/IP connections using the SMTP or LMTP protocol. The list of hosts to try can either be taken from the address that is being processed (having been set up by the router), or specified explicitly for the transport. Timeout and retry processing (see chapter 31) is applied to each IP address independently.

29.1 Multiple messages on a single connection

The sending of multiple messages over a single TCP/IP connection can arise in two ways:

- If a message contains more than **max_rcpt** (see below) addresses that are routed to the same host, more than one copy of the message has to be sent to that host. In this situation, multiple copies may be sent in a single run of the **smtp** transport over a single TCP/IP connection. (What Exim actually does when it has too many addresses to send in one message also depends on the value of the global **remote_max_parallel** option. Details are given in section 42.1.)
- 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 send multiple copies and possibly create yet another process.

For each copy sent over the same TCP/IP connection, a sequence counter is incremented, and if it ever gets to the value of **connection_max_messages**, no further messages are sent over that connection.

29.2 Use of the $host variable

At the start of a run of the **smtp** transport, the values of **$host** and **$host_address** are the name and IP address of the first host on the host list passed by the router. However, when the transport is about to connect to a specific host, and while it is connected to that host, **$host** and **$host_address** are set to the values for that host. These are the values that are in force when the **helo_data**, **hosts_try_auth**, **interface**, **serialize_hosts**, and the various TLS options are expanded.

29.3 Private options for smtp

The private options of the **smtp** transport 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**, delivery is deferred by default. However, if **allow_localhost** is set, Exim 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 port to which the message is sent).

authenticated_sender Type: *string*† Default: *unset*

This option sets a value for the AUTH= item on the outgoing MAIL command, overriding any existing authenticated sender value. If the string expansion is forced to fail, the option is ignored. Other expansion failures cause delivery to be deferred. If the result of expansion is an empty string, that is also ignored.

This option allows you to use the **smtp** transport in LMTP mode to deliver mail to Cyrus IMAP and provide the proper local part as the ‘authenticated sender’, via a setting such as:

```
authenticated_sender = $local_part
```

This removes the need for IMAP subfolders to be assigned special ACLs to allow direct delivery to those subfolders.

Because of expected uses such as that just described for Cyrus (when no domain is involved), there is no checking on the syntax of the provided value.

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: *5m*

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. However, it has been observed that on some systems there is no system timeout, which is why the default value for this option is 5 minutes, a value recommended by RFC 1123.

connection_max_messages Type: *integer* Default: *500*

This controls the maximum number of separate message deliveries that are sent over a single TCP/IP connection. If the value is zero, there is no limit. For testing purposes, this value can be overridden by the **-oB** command line option.

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. See also **final_timeout**.

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, 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 the **gethostbyname** option is false, the RES_DEFNAMES resolver option is set. See the **qualify_single** option in chapter 16 for more details.

dns_search_parents Type: *boolean* Default: *false*

If the **hosts** or **fallback_hosts** option is being used, and the **gethostbyname** option is false, the RES_DNSRCH resolver option is set. See the **search_parents** option in chapter 16 for more details.

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, which associate them with the addresses they process. As for the **hosts** option without **hosts_override**, **fallback_hosts** specified on the transport is used only if the address does not have its own associated fallback host list. Unlike **hosts**, a setting of **fallback_hosts** on an address is not overridden by **hosts_override**. However, **hosts_randomize** does apply to fallback host lists.

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** option, 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()* (or *getipnodebyname()* when available) instead of using the DNS. Of course, that function may in fact use the DNS, but it may also consult other sources of information such as **/etc/hosts**.

helo_data Type: *string*† Default: `$primary_hostname`

The value of this option is expanded, and used as the argument for the EHLO or HELO command that starts the outgoing SMTP session.

hosts Type: *string list*† Default: *unset*

Hosts are associated with an address by a router such as **dnslookup**, 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, and not all of them can provide an associated host list. The **hosts** option specifies a list of hosts which are used if the address being processed does not have any hosts associated with it. The hosts specified by **hosts** are also used, whether or not the address has its own hosts, if **hosts_override** is set.

The string is first expanded, before being interpreted as a colon-separated list of host names or IP addresses. If the expansion fails, delivery is deferred. Unless the failure was caused by the inability to complete a lookup, the error is logged to the panic log as well as the main log. Host names are looked up either by searching directly for address records in the DNS or by calling *gethostbyname()* (or *getipnodebyname()* when available), 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 IPv4 and IPv6 addresses, both types of address are used.

During delivery, the hosts are tried in order, subject to their retry status, unless **hosts_randomize** is set.

hosts_avoid_tls Type: *host list*† Default: *unset*

Exim will not try to start a TLS session when delivering to any host that matches this list. See chapter 36 for details of TLS.

hosts_max_try Type: *integer* Default: *5*

This option limits the number of IP addresses that are tried for any one delivery, in cases where attempts to connect suffer host errors. The limit does not apply, for example, to cases where temporary address errors (4*xx* responses to RCPT commands) are received. Section 29.4 describes in detail how the value of this option is used.

hosts_nopass_tls Type: *host list*† Default: *unset*

For any host that matches this list, a connection on which a TLS session has been started will not be passed to a new delivery process for sending another message on the same connection. See section 36.5 for an explanation of when this might be needed.

hosts_override Type: *boolean* Default: *false*

If this option is set and the **hosts** option is also set, any hosts that are attached to the address are ignored, and instead the hosts specified by the **hosts** option are always used. This option does not apply to **fallback_hosts**.

hosts_randomize Type: *boolean* Default: *false*

If this option is set, and either the list of hosts is taken from the **hosts** or the **fallback_hosts** option, or the hosts supplied by the router were not obtained from MX records (this includes fallback hosts from the router), and were not randomizied by the router, the order of trying the hosts is randomized each time the transport runs. Randomizing the order of a host list can be used to do crude load sharing.

When **hosts_randomize** is true, a host list may be split into groups whose order is separately randomized. This makes it possible to set up MX-like behaviour. The boundaries between groups are indicated by an item that is just + in the host list. For example:

```
hosts = host1:host2:host3:+:host4:host5
```

The order of the first three hosts and the order of the last two hosts is randomized for each use, but the first three always end up before the last two. If **hosts_randomize** is not set, a + item in the list is ignored.

hosts_require_auth Type: *host list*† Default: *unset*

This option provides a list of servers for which authentication must succeed before Exim will try to transfer a message. If authentication fails for servers which are not in this list, Exim tries to send unauthenticated. If authentication fails for one of these servers, delivery is deferred. This temporary error is detectable in the retry rules, so it can be turned into a hard failure if required. See also **hosts_try_auth**, and chapter 32 for details of authentication.

hosts_require_tls Type: *host list*† Default: *unset*

Exim will insist on using a TLS session when delivering to any host that matches this list. See chapter 36 for details of TLS.

hosts_try_auth Type: *host list*† Default: *unset*

This option provides a list of servers to which, provided they announce authentication support, Exim will attempt to authenticate as a client when it connects. If authentication fails, Exim will try to transfer the message unauthenticated. See also **hosts_require_auth**, and chapter 32 for details of authentication.

interface Type: *string list*† Default: *unset*

This option specifies which interface to bind to when making an outgoing SMTP call. The variables **$host** and **$host_address** refer to the host to which a connection is about to be made during the expansion of the string. Forced expansion failure, or an empty string result causes the option to be ignored. Otherwise, after expansion, the string must be a colon-separated list of IP addresses, for example:

```
interface = <; 192.168.123.123 ; 3ffe:ffff:836f::fe86:a061
```

The first interface of the correct type (IPv4 or IPv6) is used for the outgoing connection. If none of them are the correct type, the option is ignored. If **interface** is not set, or is ignored, the system's IP functions choose which interface to use if the host has more than one.

keepalive Type: *boolean* Default: *true*

This option controls the setting of SO_KEEPALIVE on outgoing TCP/IP socket connections. When set, it causes the kernel to probe idle connections periodically, by sending packets with 'old' sequence numbers. The other end of the connection should send a acknowledgement if the connection is still okay or a reset if the connection has been aborted. 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. The keepalive mechanism takes several hours to detect unreachable hosts.

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.

port Type: *string* Default: *see below*

This option specifies the TCP/IP port on the server to which Exim connects. If it begins with a digit it is taken as a port number; otherwise it is looked up using *getservbyname()*. The default value is normally 'smtp', but if **protocol** is set to 'lmtp', the default is 'lmtp'.

protocol Type: *string* Default: *smtp*

If this option is set to 'lmtp' instead of 'smtp', the default value for the **port** option changes to 'lmtp', and the transport operates the LMTP protocol (RFC 2033) instead of SMTP. This protocol is sometimes used for local deliveries into closed message stores. Exim also has support for running LMTP over a pipe to a local process – see chapter 27.

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. The record is 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.

If you set up this kind of serialization, you should also arrange to delete the relevant hints database whenever your system reboots. The names of the files start with **misc** and they are kept in the

spool/db directory. There may be one or two files, depending on the type of DBM in use. The same files are used for ETRN serialization.

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.

tls_certificate Type: *string*† Default: *unset*

The value of this option must be the absolute path to a file which contains the client's certificate, for use when sending a message over an encrypted connection. The values of **$host** and **$host_address** are set to the name and address of the server during the expansion. See chapter 36 for details of TLS.

tls_privatekey Type: *string*† Default: *unset*

The value of this option must be the absolute path to a file which contains the client's private key, for use when sending a message over an encrypted connection. The values of **$host** and **$host_address** are set to the name and address of the server during the expansion. If this option is unset, the private key is assumed to be in the same file as the certificate. See chapter 36 for details of TLS.

tls_require_ciphers Type: *string*† Default: *unset*

The value of this option must be a list of permitted cipher suites, for use when setting up an encrypted connection. The values of **$host** and **$host_address** are set to the name and address of the server during the expansion. See chapter 36 for details of TLS; note that this option is used in different ways by OpenSSL and GnuTLS (see section 36.1).

tls_tempfail_tryclear Type: *boolean* Default: *true*

When the server host is not in **hosts_require_tls**, and there is a problem in setting up a TLS session, this option determines whether or not Exim should try to deliver the message unencrypted. If it is set false, delivery to the current host is deferred; if there are other hosts, they are tried. If this option is set true, Exim attempts to deliver unencrypted after a 4*xx* response to STARTTLS. Also, if STARTTLS is accepted, but the subsequent TLS negotiation fails, Exim closes the current connection (because it is in an unknown state), opens a new one to the same host, and then tries the delivery in clear.

tls_verify_certificates Type: *string*† Default: *unset*

The value of this option must be the absolute path to a file or a directory containing permitted server certificates, for use when setting up an encrypted connection. The values of **$host** and **$host_address** are set to the name and address of the server during the expansion. See chapter 36 for details of TLS.

29.4 How the value of hosts_max_try is used

The **hosts_max_try** option limits the number of hosts that are tried for a single delivery. However, despite the term 'host' in its name, the option actually applies to each IP address independently. In other words, a multihomed host is treated as several independent hosts, just as it is for retrying.

Many of the larger ISPs have multiple MX records which often point to multihomed hosts. As a result, a list of a dozen or more IP addresses may be created as a result of routing one of these domains.

Trying every single IP address on such a long list does not seem sensible; if several at the top of the list fail, it is reasonable to assume there is some problem that is likely to affect all of them. Roughly

speaking, the value of **hosts_max_try** is the maximum number that are tried before deferring the delivery. However, the logic cannot be quite that simple.

Firstly, IP addresses that are skipped because their retry times have not arrived do not count, and in addition, addresses that are past their retry limits are also not counted, even when they are tried. This means that when some IP addresses are past their retry limits, more than the value of **hosts_max_retry** may be tried. The reason for this behaviour is to ensure that all IP addresses are considered before timing out an email address.

Secondly, when the **hosts_max_try** limit is reached, Exim looks down the host list to see if there is a subsequent host with a different (higher valued) MX. If there is, that host is used next, and the current IP address is used but not counted. This behaviour helps in the case of a domain with a retry rule that hardly ever delays any hosts, as is now explained:

Consider the case of a long list of hosts with one MX value, and a few with a higher MX value. If **hosts_max_try** is small (the default is 5) only a few hosts at the top of the list are tried at first. With the default retry rule, which specifies increasing retry times, the higher MX hosts are eventually tried when those at the top of the list are skipped because they have not reached their retry times.

However, it is common practice to put a fixed short retry time on domains for large ISPs, on the grounds that their servers are rarely down for very long. Unfortunately, these are exactly the domains that tend to resolve to long lists of hosts. The short retry time means that the lowest MX hosts are tried every time. The attempts may be in a different order because of random sorting, but without the special MX check mentioned about, the higher MX hosts would never be tried at all because the lower MX hosts are never all past their retry times.

With the special check, Exim tries least one address from each MX value, even if the **hosts_max_try** limit has already been reached.

30. Address rewriting

There are some circumstances in which Exim automatically rewrites domains in addresses. The two most common are when an address is given without a domain (referred to as an ‘unqualified address’) or when an address contains an abbreviated domain that is expanded by DNS lookup.

Unqualified envelope addresses are accepted only for locally submitted messages, or messages from hosts that match **sender_unqualified_hosts** or **recipient_unqualified_hosts**, respectively. Unqualified addresses in header lines are qualified if they are in locally submitted messages, or messages from hosts that are permitted to send unqualified envelope addresses. Otherwise, unqualified addresses in header lines are neither qualified nor rewritten.

One situation in which Exim does *not* rewrite a domain is when it is the name of a CNAME record in the DNS. The older RFCs suggest that such a domain should be rewritten using the ‘canonical’ name, and some MTAs do this. The new RFCs do not contain this suggestion.

This chapter is about address rewriting that is explicitly specified in the configuration. Some people believe that configured 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.

In general, rewriting addresses from your own system or domain has some legitimacy. Rewriting other addresses should be done only with great care and in special circumstances. The author of Exim believes that rewriting should be used sparingly, and mainly for ‘regularizing’ addresses in your own domains. Although it can sometimes be used as a routing tool, this is very strongly discouraged.

There are two commonly encountered circumstances where rewriting is used, as illustrated by these examples:

- The company whose domain is *hitch.fict.example* has a number of hosts that exchange mail with each other behind a firewall, but there is only a single gateway to the outer world. The gateway rewrites **.hitch.fict.example* as *hitch.fict.example* when sending mail off-site.
- A host rewrites the local parts of its own users so that, for example, *fp42@hitch.fict.example* becomes *Ford.Prefect@hitch.fict.example*.

Configured address rewriting can take place at several different stages of a message’s processing. The main rewriting happens when a message is received, but it can also happen when a new address is generated during routing (for example, by aliasing), and when a message is transported.

The rewriting rules that appear in the ‘rewrite’ section of the configuration file apply to addresses in incoming messages, and to addresses that are generated from the envelope recipients by redirection, unless **no_rewrite** is set on the relevant routers. Roughly speaking, they apply to each address the first time Exim sees it. These rules operate both on envelope addresses and on addresses in header lines. Each rule specifies to which types of address it applies.

At transport time, additional rewriting of addresses in header lines can be specified by setting the generic **headers_rewrite** option on a transport. This option contains rules that are identical in form to those in the rewrite section of the configuration file. In addition, the outgoing envelope sender can be rewritten by means of the **return_path** transport option. However, it is not possible to rewrite envelope recipients at transport time.

Rewriting of addresses in header lines applies only to those headers that were received with the message, and, in the case of transport rewriting, those that were added by a system filter. That is, it applies only to those headers that are common to all copies of the message. Header lines that are added by individual routers or transports (and which are therefore specific to individual recipient addresses) are not rewritten.

The remainder of this chapter describes the rewriting rules that are used in the main rewrite section of the configuration file, and also in the generic **headers_rewrite** option that can be set on any transport.

30.1 Testing the rewriting rules that apply on input

Exim's input rewriting configuration appears in a part of the run time configuration file headed by 'begin rewrite'. It can be tested by the **-brw** command line option. This takes an address (which can be a full RFC 2822 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 in an incoming message, that is, for each different header and for the envelope sender and recipient fields. For example,

```
exim -brw ph10@exim.workshop.example
```

might produce the output

```
    sender: Philip.Hazel@exim.workshop.example
      from: Philip.Hazel@exim.workshop.example
        to: ph10@exim.workshop.example
        cc: ph10@exim.workshop.example
       bcc: ph10@exim.workshop.example
  reply-to: Philip.Hazel@exim.workshop.example
  env-from: Philip.Hazel@exim.workshop.example
    env-to: ph10@exim.workshop.example
```

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. At the present time, there is no equivalent way of testing rewriting rules that are set for a particular transport.

30.2 Rewriting rules

The rewrite section of the configuration file consists of lines of rewriting rules in the form

<source pattern> <replacement> <flags>

Rewriting rules that are specified for the **headers_rewrite** generic transport option are given as a colon-separated list. Each item in the list takes the same form as a line in the main rewriting configuration (except that any colons must be doubled, of course).

The formats of source patterns and replacement strings are described below. Each is terminated by white space, unless enclosed in double quotes, in which case normal quoting conventions apply inside the quotes. The flags are single characters which may appear in any order. Spaces and tabs between them are ignored.

For each address that could potentially be rewritten, the rules are scanned in order, and replacements for the address from earlier rules can themselves be replaced by later rules (but see the 'q' and 'R' flags).

The order in which addresses are rewritten is undefined, may change between releases, and must not be relied on, with one exception: when a message is received, the envelope sender is always rewritten first, before any header lines are rewritten. For example, the replacement string for a rewrite of an address in *To:* must not assume that the message's address in *From:* has (or has not) already been rewritten. However, a rewrite of *From:* may assume that the envelope sender has already been rewritten.

The variables **$local_part** and **$domain** can be used in the replacement string to refer to the address that is being rewritten. Note that lookup-driven rewriting can be done by a rule of the form

```
*@*   ${lookup ...
```

where the lookup key uses **$1** and **$2** or **$local_part** and **$domain** to refer to the address that is being rewritten.

30.3 Rewriting patterns

The source pattern in a rewriting rule is any item which may appear in an address list (see section 10.13). It is in fact processed as a single-item address list, which means that it is expanded before being tested against the address.

Domains in patterns should be given in lower case. Local parts in patterns are case-sensitive. If you want to do case-insensitive matching of local parts, you can use a regular expression that starts with `^(?i)`.

After matching, the numerical variables **$1**, **$2**, etc. may be set, depending on the type of match which occurred. These can be used in the replacement string to insert portions of the incoming address. **$0** always refers to the complete incoming address. When a regular expression is used, the numerical variables are set from its capturing subexpressions. For other types of pattern they are set as follows:

- If a local part or domain starts with an asterisk, the numerical variables refer to the character strings matched by asterisks, with **$1** associated with the first asterisk, and **$2** with the second, if present. For example, if the pattern

  ```
  *queen@*.fict.example
  ```

  is matched against the address *hearts-queen@wonderland.fict.example* then

  ```
  $0 = hearts-queen@wonderland.fict.example
  $1 = hearts-
  $2 = wonderland
  ```

  Note that if the local part does not start with an asterisk, but the domain does, it is **$1** that contains the wild part of the domain.

- If the domain part of the pattern is a partial lookup, the wild and fixed parts of the domain are placed in the next available numerical variables. Suppose, for example, that the address *foo@bar.baz.example* 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.example`. Then

  ```
  $1 = foo
  $2 = bar
  $3 = baz.example
  ```

  If the address *foo@baz.example* 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.example*. If a non-wild key is matched in a partial lookup, **$2** is again set to the empty string and **$3** is set to the whole domain. For non-partial domain lookups, no numerical variables are set.

30.4 Rewriting replacements

If the replacement string for a rule is a single asterisk, addresses that match the pattern and the flags are *not* rewritten, and no subsequent rewriting rules are scanned. For example,

```
hatta@lookingglass.fict.example  *  f
```

specifies that *hatta@lookingglass.fict.example* is never to be rewritten in *From:* headers.

If the replacement string is not a single asterisk, it 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, but subsequent rules may take effect. Any other expansion failure causes the entire rewriting operation to be abandoned, and an entry written to the panic log.

30.5 Rewriting flags

There are three 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.

For rules that are part of the **headers_rewrite** generic transport option, E, F, T, and S are not permitted.

30.6 Flags specifying which headers and envelope addresses to rewrite

If none of the following flag letters, nor the 'S' flag (see section 30.7) are present, a main rewriting rule applies to all headers and to both the sender and recipient fields of the envelope, whereas a transport-time rewriting rule just applies to all headers. 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.

30.7 The SMTP-time rewriting flag

The rewrite flag 'S' specifies a rewrite of incoming envelope addresses at SMTP time, as soon as an address is received in a MAIL or RCPT command, and before any other processing; even before syntax checking. The pattern is required to be a regular expression, and it is matched against the whole of the data for the command, including any surrounding angle brackets.

This form of rewrite rule allows for the handling of addresses that are not compliant with RFCs 2821 and 2822 (for example, 'bang paths' in batched SMTP input). Because the input is not required to be a syntactically valid address, the variables **$local_part** and **$domain** are not available during the expansion of the replacement string. The result of rewriting replaces the original address in the MAIL or RCPT command.

30.8 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, 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 2822 ‘phrase’ left unchanged. For example, rewriting might change

  ```
  From: Ford Prefect <fp42@restaurant.hitch.fict.example>
  ```

  into

  ```
  From: Ford Prefect <prefectf@hitch.fict.example>
  ```

  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 line to be rewritten, the entire address is replaced, not just the working part. The replacement must be a complete RFC 2822 address, including the angle brackets if necessary. If text outside angle brackets contains a character whose value is greater than 126 or less than 32 (except for tab), the text is encoded according to RFC 2047 (assuming ISO-8859-1 encoding).

  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.

30.9 Rewriting examples

Here is an example of the two common rewriting paradigms:

```
*@*.hitch.fict.example  $1@hitch.fict.example
*@hitch.fict.example    ${lookup{$1}dbm{/etc/realnames}\
                       {$value}fail}@hitch.fict.example bctfrF
```

Note the use of ‘fail’ in the lookup expansion in the second rule, forcing the string expansion to fail if the lookup does not succeed. In this context it has the effect of leaving the original address unchanged, but Exim goes on to consider subsequent rewriting rules, if any, because 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.fict.example  *
```

were inserted before the first rule, rewriting would be suppressed for the local part *root* at any domain ending in *hitch.fict.example*.

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.fict.example  "${if !eq {$sender_host_address}{}\
                         {$1@hitch.fict.example}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

```
\N^([^!]+)!(.*)@your.domain.example$\N   $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 using this as a global rewriting rule for envelope addresses. It can

provide a backdoor method for using your system as a relay, because 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.

31. Retry configuration

The ‘retry’ section of the run time 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, temporary errors are treated as permanent. 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 because the host is down, or inaccessible because of a network problem. 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, delivery of a new message to the same host is not immediately tried, but waits for the host’s retry time to arrive. If the **retry_defer** log selector is set, 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 routing as well as to delivering, except as covered in the next paragraph. The retry rules do not distinguish between these actions. It is not possible, for example, to specify different behaviour for failures to route the domain *snark.fict.example* and failures to deliver to the host *snark.fict.example*. 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 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 routers are always run, 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 is 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 pattern must be enclosed in double quotes if it contains white space. The rules are searched in order until one is found whose pattern matches the failing host or address.

The pattern is any single item that may appear in an address list (see section 10.13). It is in fact processed as a single-item address list, which means that it is expanded before being tested against the address which has been delayed. Address list processing treats a plain domain name as if it were preceded by ‘*@’, which makes it possible for many retry rules to start with just a domain. For example,

```
lookingglass.fict.example        *  F,24h,30m;
```

provides a rule for any address in the *lookingglass.fict.example* domain, whereas

```
alice@lookingglass.fict.example  *  F,24h,30m;
```

applies only to temporary failures involving the local part **alice**.

Warning: If you use a regular expression in a routing rule, it must match a complete address, not just a domain, because that is how regular expressions work in address lists.

```
^\Nxyz\d+\.abc\.example$\N         *  G,1h,10m,2      Wrong
^\N[^@]+@xyz\d+\.abc\.example$\N   *  G,1h,10m,2      Right
```

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, suppose the MX records for *a.b.c.example* are

```
a.b.c.example  MX  5  x.y.z.example
               MX  6  p.q.r.example
               MX  7  m.n.o.example
```

and the retry rules are

```
p.q.r.example    *      F,24h,30m;
a.b.c.example    *      F,4d,45m;
```

and a delivery to the host *x.y.z.example* fails. Exim scans the retry rules. The first rule matches neither the host nor the domain, so Exim looks at the second rule. This does not match the host, but it does match the domain, so it is used to calculate the retry time for the host *x.y.z.example*. Meanwhile, Exim tries to deliver to *p.q.r.example*. If this fails, the first retry rule is used, because it matches the host.

In other words, failures to deliver to host *p.q.r.example* use the first rule to determine retry times, but for all the other hosts for the domain *a.b.c.example*, the second rule is used. The second rule is also used if routing to *a.b.c.example* suffers a temporary failure.

31.2 Retry rules for specific errors

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:

auth_failed: authentication failed when trying to send to a host in the **hosts_require_auth** list in an **smtp** transport

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 by **appendfile**

quota_<time>: quota exceeded in local delivery, and the mailbox has not been read for *<time>*. For example, *quota_4d* applies to a quota error when the mailbox has not been read for four days. **Warning**: this applies only when the mailbox consists of a single file whose access time can be checked. If you are using multi-file mailboxes, a retry rule that uses this type of error field is never matched.

The quota errors apply both to system-enforced quotas and to Exim's own quota mechanism in the **appendfile** transport. The *quota* error also applies when a local delivery is deferred because a partition is full (the ENOSPC error).

31.3 Retry rule parameters

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, which is used to increase the size of the interval at each retry.

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 unusable, 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 runner process starts delivery processes for delayed messages periodically, and these attempt new deliveries only for those addresses that have passed their next retry time. If a new message arrives for a deferred address, an immediate delivery attempt occurs only if the address has passed its retry time. In the absence of new messages, the minimum time between retries is the interval between queue runner processes. There is not much point in setting retry times of five minutes if your queue runners happen only once an hour, unless there are a significant number of incoming messages (which might be the case on a system that is sending everything to a smart host, for example).

The data in the retry hints database can be inspected by using the *exim_dumpdb* or *exim_fixdb* utility programs (see chapter 45). 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.

31.4 Retry rule examples

Here are some example retry rules:

```
alice@wonderland.fict.example quota_5d  F,7d,3h
wonderland.fict.example       quota_5d
wonderland.fict.example       *         F,1h,15m; G,2d,1h,2;
lookingglass.fict.example     *         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.example* when there is an over-quota error and the mailbox has not 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.example*; 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 has not been read for at least 5 days.

The third rule handles all other errors at *wonderland.fict.example*; retries happen every 15 minutes for an hour, then with geometrically increasing intervals until two days have passed since a delivery first failed. After the first hour there is a delay of one hour, then two hours, then four hours, and so on (this is a rather extreme example).

The fourth rule controls retries for the domain *lookingglass.fict.example*. They happen every 30 minutes for 24 hours only. 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.

31.5 Timeout of retry data

Exim timestamps the data that it writes to its retry hints database. When it consults the data during a delivery it ignores any that is older than the value set in **retry_data_expire** (default 7 days). If, for example, a host hasn't been tried for 7 days, Exim will try to deliver to it immediately a message arrives, and if that fails, it will calculate a retry time as if it were failing for the first time.

This improves the behaviour for messages routed to rarely-used hosts such as MX backups. If such a host was down at one time, and happens to be down again when Exim tries a month later, using the old retry data would imply that it had been down all the time, which is not a justified assumption.

If a host really is permanently dead, this behaviour causes a burst of retries every now and again, but only if messages routed to it are rare. It there is a message at least once every 7 days the retry data never expires.

31.6 Long-term failures

Special processing happens when an email address has been failing for so long that the cutoff time for the last algorithm is reached. For example, using the default retry rule:

```
* * F,2h,15m; G,16h,1h,1.5; F,4d,6h
```

the cutoff time is four days. Reaching the retry cutoff is independent of how long any specific message has been failing; it is the length of continuous failure for the recipient address that counts.

When the cutoff time is reached for a local delivery, or for all the IP addresses associated with a remote delivery, a subsequent delivery failure causes Exim to give up on the address, and a bounce message is generated. In order to cater for new messages that use the failing address, a next retry time is still computed from the final algorithm, and is used as follows:

For local deliveries, one delivery attempt is always made for any subsequent messages. If this delivery 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, the failing email address is bounced immediately, without a delivery attempt taking place. 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, when all the hosts for a given email address have been failing for a long time, Exim bounces rather then defers until one of the hosts' retry times is reached. Then it tries once, and bounces if that attempt fails. This behaviour ensures that few resources are wasted in repeatedly trying to deliver to a broken destination, but if the host 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 no suitable IP addresses, 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, setting **delay_after_cutoff** false means that there will be many more attempts to deliver to permanently failing IP addresses than when **delay_after_cutoff** is true.

31.7 Ultimate address timeout

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. In this situation, because some messages are successfully delivered, the ‘retry clock’ for the address 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 for a given address, a delivery is attempted for that address, even if it is not yet time, and if this delivery fails, the address is timed out. A new retry time is not computed in this case, so that other messages for the same address are considered immediately.

32. SMTP authentication

The ‘authenticators’ section of Exim’s run time configuration is concerned with SMTP authentication. This facility is an extension to the SMTP protocol, described in RFC 2554, which allows a client SMTP host to authenticate itself to a server. This is a common way for a server to recognize clients that are permitted to use it as a relay. SMTP authentication is not of relevance to the transfer of mail between servers that have no managerial connection with each other.

Very briefly, the way SMTP authentication works is as follows:

- The server advertises a number of authentication *mechanisms* in response to the client’s EHLO command.
- The client issues an AUTH command, naming a specific mechanism. The command may, optionally, contain some authentication data.
- The server may issue one or more *challenges*, to which the client must send appropriate responses. In simple authentication mechanisms, the challenges are just prompts for user names and passwords. The server does not have to issue any challenges – in some mechanisms the relevant data may all be transmitted with the AUTH command.
- The server either accepts or denies authentication.
- If authentication succeeds, the client may optionally make use of the AUTH option on the MAIL command to pass an authenticated sender in subsequent mail transactions. Authentication lasts for the remainder of the SMTP connection.
- If authentication fails, the client may give up, or it may try a different authentication mechanism, or it may try transferring mail over the unauthenticated connection.

If you are setting up a client, and want to know which authentication mechanisms the server supports, you can use Telnet to connect to port 25 (the SMTP port) on the server, and issue an EHLO command. The response to this includes the list of supported mechanisms. For example:

```
$ telnet server.example 25
Trying 192.168.34.25...
Connected to server.example.
Escape character is '^]'.
220 server.example ESMTP Exim 4.20 ...
ehlo client.example
250-server.example Hello client.example [10.8.4.5]
250-SIZE 52428800
250-PIPELINING
250-AUTH PLAIN
250 HELP
```

The second-last line of this example output shows that the server supports authentication using the PLAIN mechanism. In Exim, the different authentication mechanisms are configured by specifying *authenticator* drivers. Like the routers and transports, which authenticators are included in the binary is controlled by build-time definitions. The following are currently available, included by setting

```
AUTH_CRAM_MD5=yes
AUTH_PLAINTEXT=yes
AUTH_SPA=yes
```

in **Local/Makefile**, respectively. The first of these supports the CRAM-MD5 authentication mechanism (RFC 2195), and the second can be configured to support the PLAIN authentication mechanism (RFC 2595) or the LOGIN mechanism, which is not formally documented, but used by several MUAs. The third authenticator supports Microsoft’s *Secure Password Authentication* mechanism.

The authenticators are configured using the same syntax as other drivers (see section 6.16). If no authenticators are required, no authentication section need be present in the configuration file. Each authenticator can in principle have both server and client functions. When Exim is receiving SMTP mail, it is acting as a server; when it is sending out messages over SMTP, it is acting as a client. Authenticator configuration options are provided for use in both these circumstances.

To make it clear which options apply to which situation, the prefixes **server_** and **client_** are used on option names that are specific to either the server or the client function, respectively. Server and client functions are disabled if none of their options are set. If an authenticator is to be used for both server and client functions, a single definition, using both sets of options, is required. For example:

```
cram:
  driver = cram_md5
  public_name = CRAM-MD5
  server_secret = ${if eq{$1}{ph10}{secret1}fail}
  client_name = ph10
  client_secret = secret2
```

The **server_** option is used when Exim is acting as a server, and the **client_** options when it is acting as a client.

Descriptions of the individual authenticators are given in subsequent chapters. The remainder of this chapter covers the generic options for the authenticators, followed by general discussion of the way authentication works in Exim.

32.1 Generic options for authenticators

driver Type: *string* Default: *unset*

This option must always be set. It specifies which of the available authenticators is to be used.

public_name Type: *string* Default: *unset*

This option specifies the name of the authentication mechanism that the driver implements, and by which it is known to the outside world. These names should contain only upper case letters, digits, underscores, and hyphens (RFC 2222), but Exim in fact matches them caselessly. If **public_name** is not set, it defaults to the driver's instance name.

server_advertise_condition Type: *string*† Default: *unset*

When a server is about to advertise an authentication mechanism, the condition is expanded. If it yields the empty string, '0', 'no', or 'false', the mechanism is not advertised. See section 32.2 below for further discussion.

server_debug_print Type: *string*† Default: *unset*

If this option is set and authentication debugging is enabled (see the **-d** command line option), the string is expanded and included in the debugging output when the authenticator is run as a server. This can help with checking out the values of variables.

server_set_id Type: *string*† Default: *unset*

When an Exim server successfully authenticates a client, this string is expanded using data from the authentication, and preserved for any incoming messages in the variable **$authenticated_id**. It is also included in the log lines for incoming messages. For example, a user/password authenticator configuration might preserve the user name that was used to authenticate, and refer to it subsequently during delivery of the message.

server_mail_auth_condition Type: *string*† Default: *unset*

This option allows a server to discard authenticated sender addresses supplied as part of MAIL commands in SMTP connections that are authenticated by the driver on which it is set. The option is not used as part of the authentication process; instead its value is remembered for later use.

If **server_mail_auth_condition** is unset, addresses supplied by the AUTH option of MAIL commands are always accepted. Otherwise, when an authenticated client supplies an AUTH value on a MAIL command, the saved value of **server_mail_auth_condition** option is expanded. If the expansion fails, or yields an empty string, '0', 'no', or 'false', the AUTH address is ignored. If the expansion yields any other value, the AUTH address is retained and passed on with the message. During the expansion, the address that was supplied by the AUTH keyword is available in **$authenticated_sender**.

32.2 Authentication on an Exim server

When Exim receives an EHLO command, it advertises the public names of those authenticators that are configured as servers, subject to the following conditions:

- The client host must match **auth_advertise_hosts** (default *).
- It the **server_advertise_condition** option is set, its expansion must not yield the empty string, '0', 'no', or 'false'.

The order in which the authenticators are defined controls the order in which the mechanisms are advertised.

Some mail clients (for example, some versions of Netscape) require the user to provide a name and password for authentication whenever AUTH is advertised, even though authentication may not in fact be needed (for example, Exim may be set up to allow unconditional relaying from the client by an IP address check). You can make such clients more friendly by not advertising AUTH to them. For example, if clients on the 10.9.8.0/24 network are permitted (by the ACL that runs for RCPT) to relay without authentication, you should set

```
auth_advertise_hosts = ! 10.9.8.0/24
```

so that no authentication mechanisms are advertised to them.

The **server_advertise_condition** controls the advertisement of individual authentication mechanisms. For example, it can be used to restrict the advertisement of a patricular mechanism to encrypted connections, by a setting such as:

```
server_advertise_condition = ${if eq{$tls_cipher}{}{no}{yes}}
```

If the session is encrypted, **$tls_cipher** is not empty, and so the expansion yields 'yes', which allows the advertisement to happen.

When an Exim server receives an AUTH command from a client, it rejects it immediately if AUTH was not advertised in response to an earlier EHLO command. This is the case if

- The client host does not match **auth_advertise_hosts**; or
- No authenticators are configured with server options; or
- Expansion of **server_advertise_condition** blocked the advertising of all the server authenticators.

Otherwise, Exim runs the ACL specified by **acl_smtp_auth** in order to decide whether to accept the command. If **acl_smtp_auth** is not set, AUTH is accepted from any client host.

If AUTH is not rejected by the ACL, Exim searches its configuration for a server authentication mechanism that was advertised in response to EHLO and that matches the one named in the AUTH command. If it finds one, it runs the appropriate authentication protocol, and authentication either succeeds or fails. If there is no matching advertised mechanism, the AUTH command is rejected with a 504 error.

When a message is received from an authenticated host, the value of **$received_protocol** is set to ‘asmtp’ instead of ‘esmtp’, and **$sender_host_authenticated** contains the name (not the public name) of the authenticator driver that successfully authenticated the client from which the message was received. This variable is empty if there was no successful authentication.

32.3 Testing server authentication

Exim’s **-bh** option can be useful for testing server authentication configurations. The data for the AUTH command has to be sent using base64 encoding. A quick way to produce such data for testing is the following Perl script:

```
use MIME::Base64;
printf ("%s", encode_base64(eval "\"$ARGV[0]\""));
```

This interprets its argument as a Perl string, and then encodes it. The interpretation as a Perl string allows binary zeros, which are required for some kinds of authentication, to be included in the data. For example, a command line to run this script on such data might be

```
encode '\0user\0password'
```

Note the use of single quotes to prevent the shell interpreting the backslashes, so that they can be interpreted by Perl to specify characters whose code value is zero.

Warning 1: If either of the user or password strings starts with an octal digit, you must use three zeros instead of one after the leading backslash. If you do not, the octal digit that starts your string will be incorrectly interpreted as part of the code for the first character.

Warning 2: If there are characters in the strings that Perl interprets specially, you must use a Perl escape to prevent them being misinterpreted. For example, a command such as

```
encode '\0user@domain.com\0pas$$word'
```

gives an incorrect answer because of the unescaped ‘@’ and ‘$’ characters.

If you have the **mimencode** command installed, another way to do produce base64-encoded strings is to run the command

```
echo -e -n ‘\0user\0password’ | mimencode
```

The **-e** option of **echo** enables the interpretation of backslash escapes in the argument, and the **-n** option specifies no newline at the end of its output. However, not all versions of **echo** recognize these options, so you should check your version before relying on this suggestion.

32.4 Authenticated senders

When a client host has authenticated itself, Exim pays attention to the AUTH parameter on incoming SMTP MAIL commands. Otherwise, it accepts the syntax, but ignores the data. The data is also ignored if expansion of the **server_mail_auth_condition** option on the authenticator yields an empty string, ‘0’, ‘no’, or ‘false’.

Otherwise, unless the AUTH data is the string ‘<>’, it is set as the authenticated sender of the message. The value is available during delivery in the **$authenticated_sender** variable, and is passed on to other hosts to which Exim authenticates as a client. Do not confuse this value with **$authenticated_id**, which is a string obtained from the authentication process, and which is not usually a complete email address.

32.5 Authentication by an Exim client

The **smtp** transport has two options called **hosts_require_auth** and **hosts_try_auth**. When the **smtp** transport connects to a server that announces support for authentication, and the host matches an entry in either of these options, Exim (as a client) tries to authenticate as follows:

- For each authenticator that is configured as a client, it searches the authentication mechanisms announced by the server for one whose name matches the public name of the authenticator.

- When it finds one that matches, it runs the authenticator's client code. The variables **$host** and **$host_address** are available for any string expansions that the client might do. They are set to the server's name and IP address. If any expansion is forced to fail, the authentication attempt is abandoned, and Exim moves on to the next authenticator. Otherwise an expansion failure causes delivery to be deferred.
- If the result of the authentication attempt is a temporary error or a timeout, Exim abandons trying to send the message to the host for the moment. It will try again later. If there are any backup hosts available, they are tried in the usual way.
- If the response to authentication is a permanent error (5xx code), Exim carries on searching the list of authenticators and tries another one if possible. If all authentication attempts give permanent errors, or if there are no attempts because no mechanisms match (or option expansions force failure), what happens depends on whether the host matches **hosts_require_auth** or **hosts_try_auth**. In the first case, a temporary error is generated, and delivery is deferred. The error can be detected in the retry rules, and thereby turned into a permanent error if you wish. In the second case, Exim tries to deliver the message unauthenticated.

When Exim has authenticated itself to a remote server, it adds the AUTH parameter to the MAIL commands it sends, if it has an authenticated sender for the message. If the message came from a remote host, the authenticated sender is the one that was receiving on an incoming MAIL command, provided that the incoming connection was authenticated and the **server_mail_auth** condition allowed the authenticated sender to be retained. If a local process calls Exim to send a message, the sender address that is built from the login name and **qualify_domain** is treated as authenticated. However, if the **authenticated_sender** option is set on the **smtp** transport, it overrides the authenticated sender that was received with the message.

33. The plaintext authenticator

The **plaintext** authenticator can be configured to support the PLAIN and LOGIN authentication mechanisms, both of which transfer authentication data as plain (unencrypted) text (though base64 encoded). The use of plain text is a security risk. If you use one of these mechanisms without also making use of SMTP encryption (see chapter 36) you should not use the same passwords for SMTP connections as you do for login accounts.

33.1 Using plaintext in a server

When running as a server, **plaintext** performs the authentication test by expanding a string. It has the following options:

server_prompts Type: *string*† Default: *unset*

The contents of this option, after expansion, must be a colon-separated list of prompt strings.

server_condition Type: *string*† Default: *unset*

This option must be set in order to configure the driver as a server. Its use is described below.

The data sent by the client with the AUTH command, or in response to subsequent prompts, is base64 encoded, and so may contain any byte values when decoded. If any data is supplied with the command, it is treated as a list of NUL-separated strings which are placed in the expansion variables **$1**, **$2**, etc. If there are more strings in **server_prompts** than the number of strings supplied with the AUTH command, the remaining prompts are used to obtain more data. Each response from the client may be a list of NUL-separated strings.

Once a sufficient number of data strings have been received, **server_condition** is expanded. If the expansion is forced to fail, authentication fails. Any other expansion failure causes a temporary error code to be returned. If the result of a successful expansion is an empty string, ‘0’, ‘no’, or ‘false’, authentication fails. If the result of the expansion is ‘1’, ‘yes’, or ‘true’, authentication succeeds and the generic **server_set_id** option is expanded and saved in **$authenticated_id**. For any other result, a temporary error code is returned, with the expanded string as the error text.

Warning: If you use a lookup in the expansion to find the user’s password, be sure to make the authentication fail if the user is unknown. There are good and bad examples at the end of the next section.

33.2 The PLAIN authentication mechanism

The PLAIN authentication mechanism (RFC 2595) specifies that three strings be sent as one item of data (that is, one combined string containing two NUL separators). The data is sent either as part of the AUTH command, or subsequently in response to an empty prompt from the server.

The second and third strings are a user name and a corresponding password. Using a single fixed user name and password as an example, this could be configured as follows:

```
fixed_plain:
  driver = plaintext
  public_name = PLAIN
  server_prompts = :
  server_condition = \
    ${if and {{eq{$2}{ph10}}{eq{$3}{secret}}}{yes}{no}}
  server_set_id = $2
```

The **server_prompts** setting specifies a single, empty prompt (empty items at the end of a string list are ignored). If all the data comes as part of the AUTH command, as is commonly the case, the prompt is not used. This authenticator is advertised in the response to EHLO as

```
250-AUTH PLAIN
```

and a client host can authenticate itself by sending the command

```
AUTH PLAIN AHBoMTAAc2VjcmV0
```

As this contains three strings (more than the number of prompts), no further data is required from the client. Alternatively, the client may just send

```
AUTH PLAIN
```

to initiate authentication, in which case the server replies with an empty prompt. The client must respond with the combined data string.

The data string is base64 encoded, as required by the RFC. This example, when decoded, is '<*NUL*>ph10<*NUL*>secret', where <*NUL*> represents a zero byte. This is split up into three strings, the first of which is empty. The condition checks that the second two are 'ph10' and 'secret' respectively.

A more sophisticated instance of this authenticator could make use of the user name in **$2** to look up a password in a file or database, and maybe do an encrypted comparison (see **crypteq** in chapter 11). Here is a example of this approach, where the passwords are looked up in a DBM file. **Warning**: This is an incorrect example:

```
server_condition = \
  ${if eq{$3}{${lookup{$2}dbm{/etc/authpwd}}}{yes}{no}}
```

The expansion uses the user name (**$2**) as the key to look up a password, which it then compares to the supplied password (**$3**). Why is this example incorrect? It works fine for existing users, but consider what happens if a non-existent user name is given. The lookup fails, but as no success/failure strings are given for the lookup, it yields an empty string. Thus, to defeat the authentication, all a client has to do is to supply a non-existent user name and an empty password. The correct way of writing this test is:

```
server_condition = ${lookup{$2}dbm{/etc/authpwd}\
  {${if eq{$value}{$3}{yes}{no}}}{no}}
```

In this case, if the lookup succeeds, the result is checked; if the lookup fails, authentication fails. If **crypteq** is being used instead of **eq**, the first example is in fact safe, because **crypteq** always fails if its second argument is empty. However, the second way of writing the test makes the logic clearer.

33.3 The LOGIN authentication mechanism

The LOGIN authentication mechanism is not documented in any RFC, but is in use in a number of programs. No data is sent with the AUTH command. Instead, a user name and password are supplied separately, in response to prompts. The plaintext authenticator can be configured to support this as in this example:

```
fixed_login:
  driver = plaintext
  public_name = LOGIN
  server_prompts = User Name : Password
  server_condition = \
    ${if and {{eq{$1}{ph10}}{eq{$2}{secret}}}{yes}{no}}
  server_set_id = $1
```

Because of the way plaintext operates, this authenticator accepts data supplied with the AUTH command (in contravention of the specification of LOGIN), but if the client does not supply it (as is the case for LOGIN clients), the prompt strings are used to obtain two data items.

Some clients are very particular about the precise text of the prompts. For example, Outlook Express is reported to recognize only 'Username:' and 'Password:'. Here is an example of a LOGIN authenticator which uses those strings, and which uses the **ldapauth** expansion condition to check the user name and password by binding to an LDAP server:

```
login:
  driver = plaintext
  public_name = LOGIN
  server_prompts = Username:: : Password::
  server_condition = ${if ldapauth \
    {user="cn=${quote_ldap_dn:$1},ou=people,o=example.org" \
    pass=${quote:$2} \
    ldap://ldap.example.org/}{yes}{no}}
  server_set_id = uid=$1,ou=people,o=example.org
```

Note the use of the **quote_ldap_dn** operator to correctly quote the DN for authentication. However, the basic **quote** operator, rather than any of the LDAP quoting operators, is the correct one to use for the password, because quoting is needed only to make the password conform to the Exim syntax. At the LDAP level, the password is an uninterpreted string.

33.4 Support for different kinds of authentication

A number of string expansion features are provided for the purpose of interfacing to different ways of user authentication. These include checking traditionally encrypted passwords from **/etc/passwd** (or equivalent), PAM, Radius, **ldapauth**, and *pwcheck*. For details see section 11.6.

33.5 Using plaintext in a client

The **plaintext** authenticator has just one client option:

client_send Type: *string*† Default: *unset*

The string is a colon-separated list of authentication data strings. Each string is independently expanded before being sent to the server. The first string is sent with the AUTH command; any more strings are sent in response to prompts from the server.

Because the PLAIN authentication mechanism requires NUL (zero) bytes in the data, further processing is applied to each string before it is sent. If there are any single circumflex characters in the string, they are converted to NULs. Should an actual circumflex be required as data, it must be doubled in the string.

This is an example of a client configuration that implements the PLAIN authentication mechanism with a fixed user name and password:

```
fixed_plain:
  driver = plaintext
  public_name = PLAIN
  client_send = ^ph10^secret
```

The lack of colons means that the entire text is sent with the AUTH command, with the circumflex characters converted to NULs. A similar example that uses the LOGIN mechanism is:

```
fixed_login:
  driver = plaintext
  public_name = LOGIN
  client_send = : ph10 : secret
```

The initial colon means that the first string is empty, so no data is sent with the AUTH command itself. The remaining strings are sent in response to prompts.

34. The cram_md5 authenticator

The CRAM-MD5 authentication mechanism is described in RFC 2195. The server sends a challenge string to the client, and the response consists of a user name and the CRAM-MD5 digest of the challenge string combined with a secret string (password) which is known to both server and client. Thus, the secret is not sent over the network as plain text, which makes this authenticator more secure than **plaintext**. However, the downside is that the secret has to be available in plain text at either end.

34.1 Using cram_md5 as a server

This authenticator has one server option, which must be set to configure the authenticator as a server:

server_secret Type: *string*† Default: *unset*

When the server receives the client's response, the user name is placed in the expansion variable **$1**, and **server_secret** is expanded to obtain the password for that user. The server then computes the CRAM-MD5 digest that the client should have sent, and checks that it received the correct string. If the expansion of **server_secret** is forced to fail, authentication fails. If the expansion fails for some other reason, a temporary error code is returned to the client.

For example, the following authenticator checks that the user name given by the client is 'ph10', and if so, uses 'secret' as the password. For any other user name, authentication fails.

```
fixed_cram:
  driver = cram_md5
  public_name = CRAM-MD5
  server_secret = ${if eq{$1}{ph10}{secret}fail}
  server_set_id = $1
```

If authentication succeeds, the setting of **server_set_id** preserves the user name in **$authenticated_id**. A more tyical configuration might look up the secret string in a file, using the user name as the key. For example:

```
lookup_cram:
  driver = cram_md5
  public_name = CRAM-MD5
  server_secret = ${lookup{$1}lsearch{/etc/authpwd}{$value}fail}
  server_set_id = $1
```

Note that this expansion explicitly forces failure if the lookup fails because **$1** contains an unknown user name.

34.2 Using cram_md5 as a client

When used as a client, the **cram_md5** authenticator has two options:

client_name Type: *string*† Default: *the primary host name*

This string is expanded, and the result used as the user name data when computing the response to the server's challenge.

client_secret Type: *string*† Default: *unset*

This option must be set for the authenticator to work as a client. Its value is expanded and the result used as the secret string when computing the response.

Different user names and secrets can be used for different servers by referring to **$host** or **$host_address** in the options.

Forced failure of either expansion string is treated as an indication that this authenticator is not prepared to handle this case. Exim moves on to the next configured client authenticator. Any other expansion failure causes Exim to give up trying to send the message to the current server.

A simple example configuration of a **cram_md5** authenticator, using fixed strings, is:

```
fixed_cram:
  driver = cram_md5
  public_name = CRAM-MD5
  client_name = ph10
  client_secret = secret
```

35. The spa authenticator

The **spa** authenticator provides client support for Microsoft's *Secure Password Authentication* mechanism, which is also sometimes known as NTLM (NT LanMan). The code for client side of this authenticator was contributed by Marc Prud'hommeaux, and much of it is taken from the Samba project (**http://www.samba.org**). The code for the server side was subsequently contributed by Tom Kistner.

The mechanism works as follows:

- After the AUTH command has been accepted, the client sends an SPA authentication request based on the user name and optional domain.
- The server sends back a challenge.
- The client builds a challenge response which makes use of the user's password and sends it to the server, which then accepts or rejects it.

Encryption is used to protect the password in transit.

35.1 Using spa as a server

The **spa** authenticator has just one server option:

server_password Type: *string*† Default: *unset*

This option is expanded, and the result must be the cleartext password for the authenticating user, whose name is at this point in **$1**. For example:

```
spa:
  driver = spa
  public_name = NTLM
  server_password = ${lookup{$1}lsearch{/etc/exim/spa_clearpass}}
```

If the expansion is forced to fail, authentication fails. Any other expansion failure causes a temporary error code to be returned.

35.2 Using spa as a client

The **spa** authenticator has the following client options:

client_domain Type: *string*† Default: *unset*

This option specifies an optional domain for the authentication.

client_password Type: *string*† Default: *unset*

This option specifies the user's password, and must be set.

client_username Type: *string*† Default: *unset*

This option specifies the user name, and must be set.

Here is an example of a configuration of this authenticator for use with the mail servers at *msn.com*:

```
msn:
  driver = spa
  public_name = MSN
  client_username = msn/msn_username
  client_password = msn_plaintext_password
  client_domain = DOMAIN_OR_UNSET
```

36. Encrypted SMTP connections using TLS/SSL

Support for TLS (Transport Layer Security), otherwise known as SSL (Secure Sockets Layer), is implemented by making use of the OpenSSL library or the GnuTLS library. There is no cryptographic code in the Exim distribution itself for implementing TLS. In order to use this feature you must install OpenSSL or GnuTLS, and then build a version of Exim that includes TLS support (see section 4.5). You also need to understand the basic concepts of encryption at a managerial level, and in particular, the way that public keys, private keys, and certificates are used.

RFC 2487 defines how SMTP connections can make use of encryption. Once a connection is established, the client issues a STARTTLS command. If the server accepts this, the client and the server negotiate an encryption mechanism. If the negotiation succeeds, the data that subsequently passes between them is encrypted.

Exim also has support for legacy clients that do not use the STARTTLS mechanism. Instead, they connect to a different port on the server (usually called the 'ssmtp' port), and expect to negotiate a TLS session as soon as the connection to the server is established. The **-tls-on-connect** command line option can be used to run an Exim server in this way from *inetd*, and it can also be used to run a special daemon that operates in this manner (use **-oX** to specify the port).

Exim's ACLs can detect whether the current SMTP session is encrypted or not, and if so, what cipher suite is in use, whether the client supplied a certificate, and whether or not that certificate was verified. This makes it possible for an Exim server to deny or accept certain commands based on the encryption state.

36.1 OpenSSL vs GnuTLS

The first TLS support in Exim was implemented using OpenSSL. Support for GnuTLS followed later, when the first versions of GnuTLS were released. To build Exim to use GnuTLS, you need to set

```
USE_GNUTLS=yes
```

in Local/Makefile, in addition to

```
SUPPORT_TLS=yes
```

You must also set TLS_LIBS and TLS_INCLUDE appropriately, so that the include files and libraries for GnuTLS can be found.

There are some differences in usage when using GnuTLS instead of OpenSSL:

- The **tls_dhparam** option is ignored, because early versions of GnuTLS had no facility for varying its Diffie-Hellman parameters. I understand that this has changed, but Exim has not been updated to provide this facility.
- GnuTLS uses RSA and D-H parameters that take a substantial amount of time to compute. It is unreasonable to re-compute them for every TLS session. Therefore, Exim keeps this data in a file in its spool directory, called **gnutls-params**. The file is owned by the Exim user and is readable only by its owner. Every Exim process that start up GnuTLS reads the RSA and D-H parameters from this file. If the file does not exist, the first Exim process that needs it computes the data and writes it to a temporary file which is renamed once it is complete. It does not matter if several Exim processes do this simultaneously (apart from wasting a few resources). Once a file is in place, new Exim processes immediately start using it.

  For maximum security, the parameters that are stored in this file should be recalculated periodically, the frequency depending on your paranoia level. Arranging this is easy; just delete the file when you want new values to be computed.
- OpenSSL identifies cipher suites using hyphens as separators, for example: DES-CBC3-SHA. GnuTLS uses underscores, for example: RSA_ARCFOUR_SHA. What is more, OpenSSL complains if underscores are present in a cipher list. To make life simpler, Exim changes underscores

to hyhens for OpenSSL and hyphens to underscores for GnuTLS when processing the list of cipher suites in the **require_ciphers** option of the **smtp** transport.

- The **require_ciphers** option of the **smtp** transport operates differently. When using OpenSSL, Exim can pass the list to the library before starting the session, and the library takes care of the checking. In GnuTLS, all that can be done, as far as I know, is to check after the session has been set up that the cipher is acceptable. This makes **require_ciphers** rather less useful.

36.2 Configuring an Exim server to use TLS

When Exim has been built with TLS support, it advertises the availability of the STARTTLS command to client hosts that match **tls_advertise_hosts**, but not to any others. The default value of this option is unset, which means that STARTTLS is not advertised at all. This default is chosen because you need to set some other options in order to make TLS avaliable, and also it is sensible for systems that want to use TLS only as a client.

If a client issues a STARTTLS command and there is some configuration problem in the server, the command is rejected with a 454 error. If the client persists in trying to issue SMTP commands, all except QUIT are rejected with the error

```
554 Security failure
```

If a STARTTLS command is issued within an existing TLS session, it is rejected with a 554 error code.

To enable TLS operations on a server, you must set **tls_advertise_hosts** to match some hosts. You can, of course, set it to * to match all hosts. However, this is not all you need to do. TLS sessions to a server won't work without some further configuration at the server end.

It is rumoured that all existing clients that support TLS/SSL use RSA encryption. To make this work you need to set, in the server,

```
tls_certificate = /some/file/name
tls_privatekey = /some/file/name
```

The first file contains the server's X509 certificate, and the second contains the private key that goes with it. These files need to be readable by the Exim user, and must always be given as full path names. They can be the same file if both the certificate and the key are contained within it. If **tls_privatekey** is not set, this is assumed to be the case. The certificate file may also contain intermediate certificates that need to be sent to the client to enable it to authenticate the server's certificate.

If you do not understand about certificates and keys, please try to find a source of this background information, which is not Exim-specific. (There are a few comments below in section 36.6.)

With just these options, Exim will work as a server with clients such as Netscape. It does not require the client to have a certificate (but see below for how to insist on this). There is one other option that may be needed in other situations. If

```
tls_dhparam = /some/file/name
```

is set, the SSL library is initialized for the use of Diffie-Hellman ciphers with the parameters contained in the file. This increases the set of cipher suites that the server supports. See the command

```
openssl dhparam
```

for a way of generating this data. At present, **tls_dhparam** is used only when Exim is linked with OpenSSL. It is ignored if GnuTLS is being used.

The strings supplied for these three options are expanded every time a client host connects. It is therefore possible to use different certificates and keys for different hosts, if you so wish, by making use of the client's IP address in **$sender_host_address** to control the expansion. If a string expansion is forced to fail, Exim behaves as if the option is not set.

The variable **$tls_cipher** is set to the cipher suite that was negotiated for an incoming TLS connection. It is included in the *Received:* header of an incoming message (by default – you can, of course,

change this), and it is also included in the log line that records a message's arrival, keyed by 'X=', unless the **tls_cipher** log selector is turned off. The **encrypted** condition can be used to test for specific cipher suites in ACLs.

The ACLs that run for subsequent SMTP commands can check the name of the cipher suite and vary their actions accordingly. The cipher suite names are those used by OpenSSL. These may differ from the names used elsewhere. For example, OpenSSL uses the name DES-CBC3-SHA for the cipher suite which in other contexts is known as TLS_RSA_WITH_3DES_EDE_CBC_SHA. Check the OpenSSL documentation for more details.

36.3 Requesting and verifying client certificates

If you want an Exim server to request a certificate when negotiating a TLS session with a client, you must set either **tls_verify_hosts** or **tls_try_verify_hosts**. You can, of course, set either of them to * to apply to all TLS connections. For any host that matches one of these options, Exim requests a certificate as part of the setup of the TLS session. The contents of the certificate are verified by comparing it with a list of expected certificates. These must be available in a file or directory that is identified by **tls_verify_certificates**.

A file can contain multiple certificates, concatenated end to end. If a directory is used, each certificate must be in a separate file, with a name (or a symbolic link) of the form <*hash*>.0, where <*hash*> is a hash value constructed from the certificate. You can compute the relevant hash by running the command

```
openssl x509 -hash -noout -in /cert/file
```

where **/cert/file** contains a single certificate.

The difference between **tls_verify_hosts** and **tls_try_verify_hosts** is what happens if the client does not supply a certificate, or if the certificate does not match any of the certificates in the collection named by **tls_verify_certificates**. If the client matches **tls_verify_hosts**, the attempt to set up a TLS session is aborted, and the incoming connection is dropped. If the client matches **tls_try_verify_hosts**, the (encrypted) SMTP session continues. ACLs that run for subsequent SMTP commands can detect the fact that no certificate was verified, and vary their actions accordingly. For example, you can insist on a certificate before accepting a message for relaying, but not when the message is destined for local delivery.

When a client supplies a certificate (whether it verifies or not), the value of the Distinguished Name of the certificate is made available in the variable **$tls_peerdn** during subsequent processing of the message. Because it is often a long text string, it is not included in the log line or *Received:* header by default. You can arrange for it to be logged, keyed by 'DN=', by setting the **tls_peerdn** log selector, and you can use **received_header_text** to change the *Received:* header. When no certificate is supplied, **$tls_peerdn** is empty.

36.4 Configuring an Exim client to use TLS

The **tls_cipher** and **tls_peerdn** log selectors apply to outgoing SMTP deliveries as well as to incoming, the latter one causing logging of the server certificate's DN. The remaining client configuration for TLS is all within the **smtp** transport.

It is not necessary to set any options to have TLS work in the **smtp** transport. If Exim is built with TLS support, and TLS is advertised by a server, the **smtp** transport always tries to start a TLS session. However, this can be prevented by setting **hosts_avoid_tls** (an option of the transport) to a list of server hosts for which TLS should not be used.

If you do not want Exim to attempt to send messages unencrypted when an attempt to set up an encrypted connection fails in any way, you can set **hosts_require_tls** to a list of hosts for which encryption is mandatory. For those hosts, delivery is always deferred if an encrypted connection cannot be set up. If there are any other hosts for the address, they are tried in the usual way.

When the server host is not in **hosts_require_tls**, Exim may try to deliver the message unencrypted. It always does this if the response to STARTTLS is a 5*xx* code. For a temporary error code, or for a failure to negotiate a TLS session after a success response code, what happens is controlled by the **tls_tempfail_tryclear** option of the **smtp** transport. If it is false, delivery to this host is deferred, and other hosts (if available) are tried. If it is true, Exim attempts to deliver unencrypted after a 4*xx* response to STARTTLS, and if STARTTLS is accepted, but the subsequent TLS negotiation fails, Exim closes the current connection (because it is in an unknown state), opens a new one to the same host, and then tries the delivery unencrypted.

The **tls_certificate** and **tls_privatekey** options of the **smtp** transport provide the client with a certificate, which is passed to the server if it requests it. If the server is Exim, it will request a certificate only if **tls_verify_hosts** or **tls_try_verify_hosts** matches the client.

If **tls_verify_certificates** is set, it must name a file or directory that contains a collection of expected certificates. The client verifies the server's certificate against this collection. If **tls_require_ciphers** is set on the **smtp** transport, it must contain a list of permitted cipher suites. If either of these checks fails, delivery to the current host is abandoned, and the **smtp** transport tries to deliver to alternative hosts, if any.

All the TLS options in the **smtp** transport are expanded before use, with **$host** and **$host_address** containing the name and address of the server to which the client is connected. Forced failure of an expansion causes Exim to behave as if the relevant option were unset.

36.5 Multiple messages on the same encrypted TCP/IP connection

Exim sends multiple messages down the same TCP/IP connection by starting up an entirely new delivery process for each message, passing the socket from one process to the next. This implementation does not fit well with the use of TLS, because there is quite a lot of state information associated with a TLS connection, not just a socket identification. Passing all the state information to a new process is not feasible. Consequently, Exim shuts down an existing TLS session before passing the socket to a new process. The new process may then try to start a new TLS session, and if successful, may try to re-authenticate if AUTH is in use, before sending the next message.

The RFC is not clear as to whether or not an SMTP session continues in clear after TLS has been shut down, or whether TLS may be restarted again later, as just described. However, if the server is Exim, this shutdown and reinitialization works. It is not known which (if any) other servers operate successfully if the client closes a TLS session and continues with unencrypted SMTP, but there are certainly some that do not work. For such servers, Exim should not pass the socket to another process, because the failure of the subsequent attempt to use it would cause Exim to record a temporary host error, and delay other deliveries to that host.

To test for this case, Exim sends an EHLO command to the server after closing down the TLS session. If this fails in any way, the connection is closed instead of being passed to a new delivery process, but no retry information is recorded.

There is also a manual override; you can set **hosts_nopass_tls** on the **smtp** transport to match those hosts for which Exim should not pass connections to new processes if TLS has been used.

36.6 Certificates and all that

In order to understand fully how TLS works, you need to know about certificates, certificate signing, and certificate authorities. This is not the place to give a tutorial, especially as I do not know very much about it myself. Some helpful introduction can be found in the FAQ for the SSL addition to Apache, currently at

http://www.modssl.org/docs/2.7/ssl_faq.html#ToC24

Other parts of the *modssl* documentation are also helpful, and have links to further files. Eric Rescorla's book, *SSL and TLS*, published by Addison-Wesley (ISBN 0-201-61598-3), contains both introductory and more in-depth descriptions. Some sample programs taken from the book are available from

http://www.rtfm.com/openssl-examples/

36.7 Certificate chains

The file named by **tls_certificate** may contain more than one certificate. This is useful in the case where the certificate that is being sent is validated by an intermediate certificate which the other end does not have. Multiple certificates must be in the correct order in the file. First the host's certificate itself, then the first intermediate certificate to validate the issuer of the host certificate, then the next intermediate certificate to validate the issuer of the first intermediate certificate, and so on, until finally (optionally) the root certificate. The root certificate must already be trusted by the recipient for validation to succeed, of course, but if it's not preinstalled, sending the root certificate along with the rest makes it available for the user to install if the receiving end is a client MUA that can interact with a user.

36.8 Self-signed certificates

You can create a self-signed certificate using the *req* command provided with OpenSSL, like this:

```
openssl req -x509 -newkey rsa:1024 -keyout file1 -out file2 \
            -days 9999 -nodes
```

file1 and **file2** can be the same file; the key and the certificate are delimited and so can be identified independently. The **-days** option specifies a period for which the certificate is valid. The **-nodes** option is important: if you do not set it, the key is encrypted with a passphrase that you are prompted for, and any use that is made of the key causes more prompting for the passphrase. This is not helpful if you are going to use this certificate and key in an MTA, where prompting is not possible.

A self-signed certificate made in this way is sufficient for testing, and may be adequate for all your requirements if you are mainly interested in encrypting transfers, and not in secure identification.

However, many clients require that the certificate presented by the server be a user (also called 'leaf' or 'site') certificate, and not a self-signed certificate. In this situation, the self-signed certificate described above must be installed on the client host as a trusted root *certification authority* (CA), and the certificate used by Exim must be a user certificate signed with that self-signed certificate.

For information on creating self-signed CA certificates and using them to sign user certificates, see the *General implementation overview* chapter of the Open-source PKI book, available online at **http://ospkibook.sourceforge.net/**.

37. Access control lists

Access Control Lists (ACLs) are defined in a separate section of the run time configuration file, headed by 'begin acl'. Each ACL definition starts with a name, terminated by a colon. Here is a complete ACL section which contains just one very small ACL:

```
begin acl

small_acl:
  accept   hosts = one.host.only
```

You can have as many lists as you like in the ACL section, and the order in which they appear does not matter. The lists are self-terminating.

The majority of ACLs are used to control Exim's behaviour when it receives certain SMTP commands. This applies both to incoming TCP/IP connections, and when a local process submits a message over a pipe (using the **-bs** option). The most common use is for controlling which recipients are accepted in incoming messages. In addition, you can also define an ACL that is used to check local non-SMTP messages. The default configuration file contains an example of a realistic ACL for checking RCPT commands. This is discussed in chapter 7.

37.1 Testing ACLs

The **-bh** command line option provides a way of testing your ACL configuration locally by running a fake SMTP session with which you interact. The host *relay-test.mail-abuse.org* provides a service for checking your relaying configuration (see section 37.20 for more details).

37.2 Specifying when ACLs are used

In order to cause an ACL to be used, you have to name it in one of the relevant options in the main part of the configuration. These options are:

acl_not_smtp	ACL for non-SMTP messages
acl_smtp_auth	ACL for AUTH
acl_smtp_connect	ACL for start of SMTP connection
acl_smtp_data	ACL after DATA
acl_smtp_etrn	ACL for ETRN
acl_smtp_expn	ACL for EXPN
acl_smtp_helo	ACL for HELO or EHLO
acl_smtp_mail	ACL for MAIL
acl_smtp_rcpt	ACL for RCPT
acl_smtp_starttls	ACL for STARTTLS
acl_smtp_vrfy	ACL for VRFY

For example, if you set

```
acl_smtp_rcpt = small_acl
```

the little ACL defined above is used whenever Exim receives a RCPT command in an SMTP dialogue. The majority of policy tests on incoming messages can be done when RCPT commands arrive. A rejection of RCPT should cause the sending MTA to give up on the recipient address contained in the RCPT command, whereas rejection at other times may cause the client MTA to keep on trying to deliver the message. It is therefore recommended that you do as much testing as possible at RCPT time.

However, you cannot test the contents of the message, for example, to verify addresses in the headers, at RCPT time. Such tests have to appear in the ACL that is run after the message has been received, before the final response to the DATA command is sent. This is the ACL specified by **acl_smtp_data**. At this time, it is no longer possible to reject individual recipients. An error response should reject the entire message. Unfortunately, it is known that some MTAs do not treat hard (5*xx*) errors correctly at

this point – they keep the message on their queues and try again later, but that is their problem, though it does waste some of your resources.

The ACL test specified by **acl_smtp_connect** happens after the test specified by **host_reject_connection** (which is now an anomaly) and any TCP Wrappers testing (if configured).

The non-SMTP ACL applies to all non-interactive incoming messages, that is, it applies to batch SMTP as well as to non-SMTP messages. (Batch SMTP is not really SMTP.) This ACL is run just before the *local_scan()* function. Any kind of rejection is treated as permanent, because there is no way of sending a temporary error for these kinds of message. Many of the ACL conditions (for example, host tests, and tests on the state of the SMTP connection such as encryption and authentication) are not relevant and are forbidden in this ACL.

37.3 ACL return codes

The result of running an ACL is either 'accept' or 'deny', or, if some test cannot be completed (for example, if a database is down), 'defer'. These results cause 2*xx*, 5*xx*, and 4*xx* return codes, respectively, to be used in the SMTP dialogue. A fourth return, 'error', occurs when there is an error such as invalid syntax in the ACL. This also causes a 4*xx* return code.

The ACLs that are relevant to message reception may also return 'discard'. This has the effect of 'accept', but causes either the entire message or an individual recipient address to be discarded. In other words, it is a blackholing facility. Use it with great care.

If the ACL for MAIL returns 'discard', all recipients are discarded, and no ACL is run for subsequent RCPT commands. The effect of 'discard' in a RCPT ACL is to discard just the one address. If there are no recipients left when the message's data is received, the DATA ACL is not run. A 'discard' return from the DATA or the non-SMTP ACL discards all the remaining recipients.

The *local_scan()* function is always run, even if there are no remaining recipients; it may create new recipients.

37.4 Unset ACL options

The default actions when any of the **acl_smtp_***xxx* options are unset are not all the same.

For **acl_not_smtp**, **acl_smtp_auth**, **acl_smtp_connect**, **acl_smtp_data**, **acl_smtp_helo**, **acl_smtp_mail**, and **acl_smtp_starttls**, the default action is 'accept'.

For the others (**acl_smtp_etrn**, **acl_smtp_expn**, **acl_smtp_rcpt**, and **acl_smtp_vrfy**), the default action is 'deny'. This means that **acl_smtp_rcpt** must be defined in order to receive any messages over an SMTP connection. For an example, see the ACL in the default configuration file.

37.5 Data for message ACLs

When an ACL for MAIL, RCPT, or DATA is being run, the variables that contain information about the host and the message's sender (for example, **$sender_host_address** and **$sender_address**) are set, and can be used in ACL statements. In the case of RCPT (but not MAIL or DATA), **$domain** and **$local_part** are set from the argument address.

The **$message_size** variable is set to the value of the SIZE parameter on the MAIL command at MAIL and RCPT time, or -1 if that parameter was not given. Its value is updated to the true message size by the time the ACL after DATA is run.

The **$rcpt_count** variable increases by one for each RCPT command received. The **$recipients_count** variable increases by one each time a RCPT command is accepted, so while an ACL for RCPT is being processed, it contains the number of previously accepted recipients. At DATA time, **$rcpt_count** contains the total number of RCPT commands, and **$recipients_count** contains the total number of accepted recipients.

37.6 Data for non-message ACLs

When an ACL for AUTH, ETRN, EXPN, STARTTLS, or VRFY is being run, the remainder of the SMTP command line is placed in **$smtp_command_argument**. This can be tested using a **condition** condition. For example, here is an ACL for use with AUTH, which insists that either the session is encrypted, or the CRAM-MD5 authentication method is used. In other words, it does not permit authentication methods that use cleartext passwords on unencrypted connections.

```
acl_check_auth:
  accept encrypted = *
  accept condition = ${if eq{${uc:$smtp_command_argument}}\
                     {CRAM-MD5}{yes}{no}}
  deny   message   = TLS encryption or CRAM-MD5 required
```

(Another way of applying this restriction is to arrange for the authenticators that use cleartext passwords not to be advertised when the connection is not encrypted. You can use the generic **server_advertise_condition** authenticator option to do this.)

37.7 Use of the ACL selection options

The value of an **acl_smtp_***xxx* option is expanded before use, so you can use different ACLs in different circumstances, and in fact the resulting string does not have to be the name of a configured list. Having expanded the string, Exim searches for an ACL as follows:

- If the string begins with a slash, Exim attempts to open the file and read its contents as an ACL. Blank lines are ignored, and lines whose first non-whitespace character is '#' are also ignored. However, continuation lines are not supported. If the file does not exist or cannot be read, an error occurs (typically causing a temporary failure of whatever caused the ACL to be run). For example:

  ```
  acl_smtp_data = /etc/acls/\
    ${lookup{$sender_host_address}lsearch\
    {/etc/acllist}{$value}{default}}
  ```

  This looks up an ACL file to use on the basis of the host's IP address, falling back to a default if the lookup fails. If an ACL is successfully read from a file, it is retained in memory for the duration of the Exim process, so that it can be re-used without actually having to re-read the file.

- If the string does not start with a slash, and does not contain any spaces, Exim searches the ACL section of the configuration for a list whose name matches the string.

- If no named ACL is found, or if the string contains spaces, Exim parses the string as an inline ACL. This can save typing in cases where you just want to have something like

  ```
  acl_smtp_vrfy = accept
  ```

  in order to allow free use of the VRFY command.

37.8 Format of an ACL

An individual ACL consists of a number of statements. Each statement starts with a verb, optionally followed by a number of conditions and other modifiers. If all the conditions are met, the verb is obeyed. If there are no conditions, the verb is always obeyed. What happens if any of the conditions are not met depends on the verb (and in one case, on a special modifier). Not all the conditions make sense at every testing point. For example, you cannot test a sender address in the ACL that is run for a VRFY command.

The verbs are as follows:

- **accept**: If all the conditions are met, the ACL returns 'accept'. If any of the conditions are not met, what happens depends on whether **endpass** appears among the conditions (for syntax see below). If the failing condition precedes **endpass**, control is passed to the next ACL statement; if

it follows **endpass**, the ACL returns ‘deny’. Consider this statement, used to check a RCPT command:

```
accept domains = +local_domains
       endpass
       verify = recipient
```

If the recipient domain does not match the **domains** condition, control passes to the next statement. If it does match, the recipient is verified, and the command is accepted if verification succeeds. However, if verification fails, the ACL yields ‘deny’, because the failing condition is after **endpass**.

- **defer**: If all the conditions are met, the ACL returns ‘defer’ which, in an SMTP session, causes a 4*xx* response to be given. For a non-SMTP ACL, **defer** is the same as **deny**, because there is no way of sending a temporary error. For a RCPT command, **defer** is much the same as using a **redirect** router and `:defer:` while verifying, but the **defer** verb can be used in any ACL, and even for a recipient it might be a simpler approach.

- **deny**: If all the conditions are met, the ACL returns ‘deny’. If any of the conditions are not met, control is passed to the next ACL statement. For example,

  ```
  deny dnslists = blackholes.mail-abuse.org
  ```

  rejects commands from hosts that are on a DNS black list.

- **discard**: This verb behaves like **accept**, except that it returns ‘discard’ from the ACL. It is permitted only on ACLs that are concerned with receiving messages, and it causes recipients to be discarded. If used in an ACL for RCPT, just the one recipient is discarded; if used for MAIL, DATA or in the non-SMTP ACL, all the message’s recipients are discarded. Recipients that are discarded before DATA do not appear in the log line when the **log_recipients** log selector is set.

- **drop**: This verb behaves like **deny**, except that an SMTP connection is forcibly closed after the 5*xx* error message has been sent. For example:

  ```
  drop   message   = I don't take more than 20 RCPTs
         condition = ${if > {$rcpt_count}{20}{yes}{no}}
  ```

  There is no difference between **deny** and **drop** for the connect-time ACL. The connection is always dropped after sending a 550 response.

- **require**: If all the conditions are met, control is passed to the next ACL statement. If any of the conditions are not met, the ACL returns ‘deny’. For example, when checking a RCPT command,

  ```
  require verify = sender
  ```

  passes control to subsequent statements only if the message’s sender can be verified. Otherwise, it rejects the command.

- **warn**: If all the conditions are met, some warning action is taken. In all cases, control passes to the next ACL statement. In the case of testing an incoming message, the warning action consists of adding header lines to the message, and/or writing an entry in the main log. Additional header lines are specified by the **message** modifier, as in this example:

  ```
  warn message = X-blacklisted-at: $dnslist_domain
       dnslists = blackholes.mail-abuse.org : \
                  dialup.mail-abuse.org
  ```

  If an identical header line is requested several times (provoked, for example, by multiple RCPT commands), only one copy is actually added to the message. Header lines that are added by an ACL at MAIL or RCPT time are not visible in string expansions in the ACL for subsequent RCPT commands. However they are are visible in string expansions in the ACL that is run after DATA. If you want to preserve data between MAIL and RCPT ACLs, you can use ACL variables, as described in the next section.

Log lines are specified by the **log_message** modifier. For ACLs that are not concerned with incoming messages, only the logging action is available.

If any condition on a **warn** statement cannot be completed (that is, there is some sort of defer), the warning action does not take place. The incident is logged, but the ACL continues to be processed.

At the end of each ACL there is an implicit unconditional **deny**.

As you can see from the examples above, the conditions and modifiers are written one to a line, with the first one on the same line as the verb, and subsequent ones on following lines. If you have a very long condition, you can continue it onto several physical lines by the usual \ continuation mechanism. It is conventional to align the conditions vertically.

37.9 ACL variables

There are some special variables that can be set explicitly during ACL processing. They can be used to pass information between different ACLs and different invocations of the same ACL in the same SMTP connection. There are two sets of these variables:

- The values of **$acl_c0** to **$acl_c9** persist throughout the SMTP connection. They are never reset.
- The values of **$acl_m0** to **$acl_m9** persist only while a message is being received. They are reset afterwards. They are also reset by MAIL, RSET, EHLO, HELO, and after starting up a TLS session. However, they still contain their values when the *local_scan()* function is run.

The ACL variables are set by modifier called **set**. For example:

```
accept hosts = whatever
       set acl_m4 = some value
```

Note that the leading dollar sign is not used when naming a variable that is to be set. If you want to set a variable without taking any action, you can use a **warn** verb without any other modifiers.

As their name suggests, these variables are set only during ACL processing. At all other times they are empty.

37.10 Condition and modifier processing

An exclamation mark preceding a condition negates its result. For example,

```
deny   domains = *.dom.example
      !verify = recipient
```

causes the ACL to return ‘deny’ if the recipient domain ends in *dom.example*, but the recipient address cannot be verified.

The arguments of conditions and modifiers are expanded. A forced failure of an expansion causes a condition to be ignored, that is, it behaves as if the condition is true. Consider these two statements:

```
accept  senders = ${lookup{$host_name}lsearch\
                  {/some/file}{$value}fail}
accept  senders = ${lookup{$host_name}lsearch\
                  {/some/file}{$value}{}}
```

Each attempts to look up a list of acceptable senders. If the lookup succeeds, the returned list is searched, but if the lookup fails the behaviour is different in the two cases. The **fail** in the first statement causes the condition to be ignored, leaving no further conditions. The **accept** verb therefore succeeds. The second statement, however, generates an empty list when the lookup fails. No sender can match an empty list, so the condition fails and therefore the **accept** also fails.

ACL modifiers appear mixed in with conditions in ACL statements. They specify actions that are taken as the conditions for a statement are checked, or text for messages that are used when access is denied or a warning is generated.

The positioning of the modifiers in an ACL statement important, because the processing of a verb ceases as soon as its outcome is known. Only those modifiers that have already been encountered will take effect. For the **accept** and **require** statements, this means that processing stops as soon as a false condition is met. For example, consider this use of the **message** modifier:

```
require message = Can't verify sender
        verify = sender
        message = Can't verify recipient
        verify = recipient
        message = This message cannot be used
```

If sender verification fails, Exim knows that the result of the statement is 'deny', so it goes no further. The first **message** modifier has been seen, so its text is used as the error message. If sender verification succeeds, but recipient verification fails, the second message is used. If recipient verification succeeds, the third message becomes 'current', but is never used because there are no more conditions to cause failure.

For the **deny** verb, on the other hand, it is always the last **message** modifier that is used, because all the conditions must be true for rejection to happen. Specifying more than one **message** modifier does not make sense, and the message can even be specified after all the conditions. For example:

```
deny   hosts = ...
      !senders = *@my.domain.example
       message = Invalid sender from client host
```

The 'deny' result does not happen until the end of the statement is reached, by which time Exim has set up the message.

37.11 ACL modifiers

The ACL modifiers are as follows:

control = <*text*>

This modifier may appear only in ACLs for commands relating to incoming messages. It affects the subsequent processing of the message, provided that the message is eventually accepted. The text must be one of the words 'freeze' or 'queue_only'. These cause the message to be frozen or just queued (without immediate delivery), respectively. Once one of these controls is set, it remains set for the message. For example, if **control** is used in a RCPT ACL, it applies to the whole message, not just the individual recipient. The **control** modifier can be used in several different ways. For example:

- It can be at the end of an **accept** statement:

  ```
  accept  ...some conditions...
          control = queue_only
  ```

  In this case, the control is applied when this statement yields 'accept'.

- It can be in the middle of an **accept** statement:

  ```
  accept  ...some conditions...
          control = queue_only
          ...some more conditions...
  ```

  If the first set of conditions are true, the control is applied, even if the statement does not accept because one of the second set of conditions is false. In this case, some subsequent statement must yield 'accept' for the control to be relevant.

- It can be used with **warn** to apply the control, leaving the decision about accepting or denying to a subsequent verb. For example:

```
warn    ...some conditions...
        control = freeze
accept  ...
```

Note: If neither **message** nor **log_message** is set for a **warn** statement, it does not add anything to the message and does not write a log entry.

delay = <*time*>

This modifier causes Exim to wait for the time interval before proceeding. The time is given in the usual Exim notation. This modifier may appear in any ACL. The delay happens as soon as the modifier is processed. Like **control**, **delay** can be used with **accept** or **deny**, for example:

```
deny    ...some conditions...
        delay = 30s
```

The delay happens if all the conditions are true, before the statement returns ‘deny’. Compare this with:

```
deny    delay = 30s
        ...some conditions...
```

which waits for 30s before processing the conditions. The **delay** modifier can also be used with **warn** and together with **control**:

```
warn    ...some conditions...
        delay = 2m
        control = freeze
accept  ...
```

endpass

This modifier, which has no argument, is recognized only in **accept** statements. It marks the boundary between the conditions whose failure causes control to pass to the next statement, and the conditions whose failure causes the ACL to return ‘deny’. See the description of **accept** above.

log_message = <*text*>

This modifier sets up a message that is used as part of the log message if the ACL denies access. For example:

```
require log_message = wrong cipher suite $tls_cipher
        encrypted   = DES-CBC3-SHA
```

log_message adds to any underlying error message that may exist because of the condition failure. For example, while verifying a recipient address, a *:fail:* redirection might have already set up a message. Although the message is usually defined before the conditions to which it applies, the expansion does not happen until Exim decides that access is to be denied. This means that any variables that are set by the condition are available for inclusion in the message. For example, the **$dnslist_**<*xxx*> variables are set after a DNS black list lookup succeeds. If the expansion of **log_message** fails, or if the result is an empty string, the modifier is ignored.

If **log_message** is used with a **warn** statement, ‘Warning:’ is added to the start of the logged message.

If **log_message** is not present and there is no underlying error message (for example, from the failure of address verification), but **message** is present, the **message** text is used for logging rejections. However, if it contains newlines, only the first line of the text is logged. In the absence of both **log_message** and **message**, a default built-in message is used.

message = <*text*>

This modifier sets up a text string that is expanded and used as an error message if the current statement causes the ACL to deny access. The expansion happens at the time Exim decides that access is to be denied, not at the time it processes **message**. If the expansion fails, or generates an empty string, the modifier is ignored. For ACLs that are triggered by SMTP commands, the message is returned as part of the SMTP error response.

The **message** modifier is also used with the **warn** verb to specify one or more header lines to be added to an incoming message when all the conditions are true. For non-message ACLs, **message** has no effect with **warn**.

The text is literal; any quotes are taken as literals, but because the string is expanded, backslash escapes are processed anyway. If the message contains newlines, this gives rise to a multi-line SMTP response. Like **log_message**, the contents of **message** are not expanded until after a condition has failed.

If **message** is used on a statement that verifies an address, the message specified overrides any message that is generated by the verification process. However, the original message is available in the variable **$acl_verify_message**, so you can incorporate it into your message if you wish. In particular, if you want the text from **:fail:** items in **redirect** routers to be passed back as part of the SMTP response, you should either not use a **message** modifier, or make use of **$acl_verify_message**.

set <*acl_name*> = <*value*>

This modifier puts a value into one of the ACL variables (see section 37.9).

37.12 ACL conditions

Not all conditions are relevant in all circumstances. For example, testing senders and recipients does not make sense in an ACL that is being run as the result of the arrival of an ETRN command, and checks on message headers can be done only in the ACLs specified by **acl_smtp_data** and **acl_not_smtp**.

The conditions are:

acl = <*name of acl or ACL string or file name* >

The possible values of the argument are the same as for the **acl_smtp_***xxx* options. The named or inline ACL is run. If it returns ‘accept’ the condition is true; if it returns ‘deny’ the condition is false; if it returns ‘defer’, the current ACL returns ‘defer’. ACLs may be nested up to 20 deep; the limit exists purely to catch runaway loops.

This condition allows you to use different ACLs in different conditions. For example, different ACLs can be used to handle RCPT commands for different local users or different local domains.

authenticated = <*string list*>

If the SMTP connection is not authenticated, the condition is false. Otherwise, the name of the authenticator is tested against the list. To test for authentication by any authenticator, you can set

```
authenticated = *
```

condition = <*string*>

This feature allows you to make up custom conditions. If the result of expanding the string is an empty string, the number zero, or one of the strings ‘no’ or ‘false’, the condition is false. If the result is any non-zero number, or one of the strings ‘yes’ or ‘true’, the condition is true. For any other values, some error is assumed to have occured, and the ACL returns ‘defer’.

dnslists = *<list of domain names and other data>*

This condition checks for entries in DNS black lists. These are also known as ‘RBL lists’, after the original Realtime Blackhole List, but note that the use of the lists at *mail-abuse.org* now carries a charge. In its simplest form, the **dnslists** condition tests whether the calling host is on a DNS black list by looking up the inverted IP address in one or more DNS domains. For example, if the calling host’s IP address is 192.168.62.43, and the ACL statement is

```
deny dnslists = blackholes.mail-abuse.org : \
                dialups.mail-abuse.org
```

the following domains are looked up:

```
43.62.168.192.blackholes.mail-abuse.org
43.62.168.192.dialups.mail-abuse.org
```

If a DNS lookup times out or otherwise fails to give a decisive answer, Exim behaves as if the host is not on the relevant list. This is usually the required action when **dnslists** is used with **deny** (which is the most common usage), because it prevents a DNS failure from blocking mail. However, you can change this behaviour by putting one of the following special items in the list:

`+include_unknown`	behave as if the item is on the list
`+exclude_unknown`	behave as if the item is not on the list (default)
`+defer_unknown`	give a temporary error

Each of these applies to any subsequent items on the list. For example:

```
deny dnslists = +defer_unknown : foo.bar.example
```

Testing the list of domains stops as soon as a match is found. If you want to warn for one list and block for another, you can use two different statements:

```
deny  dnslists = blackholes.mail-abuse.org
warn  dnslists = dialups.mail-abuse.org
```

There are some lists which are keyed on domain names rather than inverted IP addresses (see for example the *domain based zones* link at **http://www.rfc-ignorant.org/**). You can change the name that is looked up by adding additional data to a **dnslists** item, introduced by a slash. For example,

```
deny  message  = Sender’s domain is listed at $dnslist_domain
      dnslists = dsn.rfc-ignorant.org/$sender_address_domain
```

This particular example is useful only in ACLs that are obeyed after the RCPT or DATA commands, when a sender address is available. If (for example) the message’s sender is *user@tld.example* the name that is looked up is

```
tld.example.dsn.rfc-ignorant.org
```

You can mix entries with and without additional data in the same **dnslists** condition.

When an entry is found in a DNS black list, the variable **$dnslist_domain** contains the name of the domain which matched, **$dnslist_value** contains the data from the entry, and **$dnslist_text** contains the contents of any associated TXT record. You can use these variables in **message** or **log_message** modifiers – although these appear before the condition in the ACL, they are not expanded until after it has failed. For example:

```
deny    hosts = !+local_networks
        message = $sender_host_address is listed \
                  at $dnslist_domain
        dnslists = rbl-plus.mail-abuse.example
```

DNS black list lookups are cached by Exim for the duration of the SMTP session, so a lookup based on the IP address is done at most once for any incoming connection. Exim does not share information between multiple incoming connections (but your local name server cache should be active).

DNS black lists are constructed using address records in the DNS. The original RBL just used the address 127.0.0.1 on the right hand side of the records, but the RBL+ list and some other lists use a number of values with different meanings. The values used on the RBL+ list are:

127.1.0.1 RBL
127.1.0.2 DUL
127.1.0.3 DUL and RBL
127.1.0.4 RSS
127.1.0.5 RSS and RBL
127.1.0.6 RSS and DUL
127.1.0.7 RSS and DUL and RBL

If you add an equals sign and an IP address after a **dnslists** domain name, you can restrict its action to DNS records with a matching right hand side. For example,

```
deny dnslists = rblplus.mail-abuse.org=127.0.0.2
```

rejects only those hosts that yield 127.0.0.2. More than one address may be given, using a comma as a separator. These are alternatives – if any one of them matches, the RBL entry operates. If there are no addresses, any address record is considered to be a match.

If you want to specify a constraining address and also change the name that is looked up, the address list must be specified first. For example:

```
deny dnslists = dsn.rfc-ignorant.org\
                =127.0.0.2/$sender_address_domain
```

domains = <*domain list*>

This condition is relevant only after a RCPT command. It checks that the domain of the recipient address is in the domain list. If percent-hack processing is enabled, it is done before this test is done. If the check succeeds with a lookup, the result of the lookup is placed in **$domain_data** until the next **domains** test.

encrypted = <*string list*>

If the SMTP connection is not encrypted, the condition is false. Otherwise, the name of the cipher suite in use is tested against the list. To test for encryption without testing for any specific cipher suite(s), set

```
encrypted = *
```

hosts = < *host list*>

This condition tests that the calling host matches the host list. If you have name lookups or wildcarded host names and IP addresses in the same host list, you should normally put the IP addresses first. For example, you could have:

```
accept hosts = 10.9.8.7 : dbm;/etc/friendly/hosts
```

The reason for this lies in the left-to-right way that Exim processes lists. It can test IP addresses without doing any DNS lookups, but when it reaches an item that requires a host name, it fails if it cannot find a host name to compare with the pattern. If the above list is given in the opposite order, the **accept** statement fails for a host whose name cannot be found, even if its IP address is 10.9.8.7.

If you really do want to do the name check first, and still recognize the IP address even if the name lookup fails, you can rewrite the ACL like this:

```
accept hosts = dbm;/etc/friendly/hosts
accept hosts = 10.9.8.7
```

The default action on failing to find the host name is to assume that the host is not in the list, so the first **accept** statement fails. The second statement can then check the IP address.

If a **hosts** condition is satisfied by means of a lookup, the result of the lookup is made available in the **$host_data** variable. This allows you, for example, to set up a statement like this:

```
deny  hosts = net-lsearch;/some/file
      message = $host_data
```

which gives a custom error message for each denied host.

local_parts = <*local part list*>

This condition is relevant only after a RCPT command. It checks that the local part of the recipient address is in the list. If percent-hack processing is enabled, it is done before this test. If the check succeeds with a lookup, the result of the lookup is placed in **$local_part_data** until the next **local_parts** test.

recipients = <*address list*>

This condition is relevant only after a RCPT command. It checks the entire recipient address against a list of recipients.

sender_domains = <*domain list*>

This condition tests the domain of the sender of the message against the given domain list.

senders = <*address list*>

This condition tests the sender of the message against the given list. To test for a bounce message, which has an empty sender, set

```
senders = :
```

verify = **certificate**

This condition is true in an SMTP session if the session is encrypted, and a certificate was received from the client, and the certificate was verified. The server requests a certificate only if the client matches **tls_verify_hosts** or **tls_try_verify_hosts** (see chapter 36).

verify = **header_sender**/<*options*>

This condition is relevant only in an ACL that is run after a message has been received, that is, in an ACL specified by **acl_smtp_data**. It checks that there is a verifiable sender address in at least one of the *Sender:*, *Reply-To:*, or *From:* header lines. Details of address verification and the options are given in the next section. You can combine this condition with the **senders** condition to restrict it to bounce messages only:

```
deny    senders = :
        message = A valid sender header is required for bounces
       !verify  = header_sender
```

verify = **header_syntax**

This condition is relevant only in an ACL that is run after a message has been received, that is, in an ACL specified by **acl_smtp_data** or **acl_not_smtp**. It checks the syntax of all header lines that can contain lists of addresses (*Sender:*, *From:*, *Reply-To:*, *To:*, *Cc:*, and *Bcc:*). Unqualified addresses (local parts without domains) are permitted only in locally generated messages and from hosts that match **sender_unqualified_hosts** or **recipient_unqualified_hosts**, as appropriate.

Note that this condition is a syntax check only. However, a common spamming ploy is to send syntactically invalid headers such as

```
To: @
```

and this condition can be used to reject such messages.

verify = helo

This condition is true if a HELO or EHLO command has been received from the client host, and its contents have been verified. Verification of these commands does not happen by default. See the description of the **helo_verify_hosts** and **helo_try_verify_hosts** options for details of how to request it.

verify = recipient/<*options*>

This condition is relevant only after a RCPT command. It verifies the current recipient. Details of address verification are given in the next section. After a recipient has been verified, the value of **$address_data** is the last value that was set while routing the address. This applies even if the verification fails. When an address that is being verified is redirected to a single address, verification continues with the new address, and in that case, the subsequent value of **$address_data** is the value for the child address.

verify = reverse_host_lookup

This condition ensures that a verified host name has been looked up from the IP address of the client host. (This may have happened already if the host name was needed for checking a host list, or if the host matched **host_lookup**.) Verification ensures that the host name obtained from a reverse DNS lookup, or one of its aliases, does, when it is itself looked up in the DNS, yield the original IP address.

If this condition is used for a locally generated message (that is, when there is no client host involved), it always succeeds.

verify = sender/<*options*>

This condition is relevant only after a MAIL or RCPT command, or after a message has been received (the **acl_smtp_data** or **acl_not_smtp** ACLs). If the message's sender is empty (that is, this is a bounce message), the condition is true. Otherwise, the sender address is verified. Details of verification are given in the next section. Exim caches the result of sender verification, to avoid doing it more than once per message.

verify = sender=address/<*options*>

This is a variation of the previous option, in which a modified address is verified as a sender.

37.13 Address verification

Several of the **verify** conditions described in the previous section cause addresses to be verified. These conditions can be followed by options that modify the verification process. The options are separated from the keyword and from each other by slashes, and some of them contain parameters. For example:

```
verify = sender/callout
verify = recipient/defer_ok/callout=10s,defer_ok
```

The first stage of verification is to run the address through the routers, in 'verify mode'. Routers can detect the difference between verification and routing for delivery, and their actions can be varied by a number of generic options such as **verify** and **verify_only** (see chapter 14).

If there is a defer error while doing this verification routing, the ACL normally returns 'defer'. However, if you include **defer_ok** in the options, the condition is forced to be true instead.

37.14 Callout verification

For non-local addresses, routing verifies the domain, but is unable to do any checking of the local part. There are situations where some means of verifying the local part is desirable. One way this can be done is to make an SMTP *callback* to the sending host (for a sender address) or a *callforward* to a subsequent host (for a recipient address), to see if the host accepts the address. We use the term *callout* to cover both cases. This facility should be used with care, because it can add a lot of resource usage

to the cost of verifying an address. However, Exim does cache the results of callouts, which helps to reduce the cost. Details are in the next section.

A successful callout does not guarantee that a real delivery to the address would succeed; on the other hand, a failing callout does guarantee that it would fail.

If the **callout** option is present on a condition that verifies an address, a second stage of verification occurs if the address is successfully routed to one or more remote hosts. The usual case is routing by a **dnslookup** or a **manualroute** router, where the router specifies the hosts. However, if a router that does not set up hosts routes to an **smtp** transport with a **hosts** setting, the transport's hosts are used. If an **smtp** transport transport has **hosts_override** set, its hosts are always used, whether or not the router supplies a host list.

When a host list is available, Exim makes SMTP connections to the remote hosts, to test whether a bounce message could be delivered to a sender address, or whether the current message could be delivered to a recipient address. For a sender address, it sends:

```
HELO <primary host name>
MAIL FROM:<>
RCPT TO:<the address to be tested>
QUIT
```

For a recipient address, the MAIL command contains the sender address of the incoming message. If the response to the RCPT command is a 2*xx* code, the verification succeeds. If it is 5*xx*, the verification fails. For any other condition, Exim tries the next host, if any. If there is a problem with all the remote hosts, the ACL yields 'defer', unless the **defer_ok** parameter of the **callout** option is given, in which case the condition is forced to succeed.

For SMTP callout connections, the port to connect to and the outgoing interface are taken from the transport to which address was routed, if it is a remote transport. Otherwise port 25 is used, and the interface is not specified.

37.15 Additional parameters for callouts

The **callout** option can be followed by an equals sign and a number of optional parameters, separated by commas. For example:

```
verify = recipient/callout=10s,defer_ok
```

The old syntax, which had **callout_defer_ok** and **check_postmaster** as separate verify options, is retained for backwards compatibility, but is now deprecated. The additional parameters for **callout** are as follows:

- *<a time>*: This specifies the timeout that applies for the callout attempt to each host. For example:

  ```
  verify = sender/callout=5s
  ```

  The default is 30 seconds. The timeout is used for each response from the remote host.

- **defer_ok**: Failure to contact any host, or any other kind of temporary error is treated as success by the ACL. However, the cache is not updated in this circumstance.

- **no_cache**: The callout cache is neither read not updated.

- **postmaster**: A successful callout check is followed by a similar check for the local part *postmaster* at the same domain. If this address is rejected, the callout fails. The result of the postmaster check is recorded in a cache record; if it is a failure, this is used to fail subsequent callouts for the domain without a connection being made, until the cache record expires.

- **random**: Before doing the normal callout check, Exim does a check for a 'random' local part at the same domain. The local part is not really random – it is defined by the expansion of the option **callout_random_local_part**, which defaults to

  ```
  $primary_host_name-$tod_epoch-testing
  ```

The idea here is to try to determine whether the remote host accepts all local parts without checking. If it does, there is no point in doing callouts for specific local parts. If the 'random' check succeeds, the result is saved in a cache record, and used to force the current and subsequent callout checks to succeed without a connection being made, until the cache record expires.

37.16 Callout caching

Exim caches the results of callouts in order to reduce the amount of resources used, unless you specify the **no_cache** parameter with the **callout** option. A hints database called 'callout' is used for the cache. Two different record types are used: one records the result of a callout check for a specific address, and the other records information that applies to the entire domain (for example, that it accepts the local part *postmaster*).

When an original callout fails, a detailed SMTP error message is given about the failure. However, for subsequent failures use the cache data, this message is not available.

The expiry times for negative and positive address cache records are independent, and can be set by the global options **callout_negative_expire** (default 2h) and **callout_positive_expire** (default 24h), respectively.

If a host gives a negative response to an SMTP connection, or rejects any commands up to and including

```
MAIL FROM:<>
```

any callout attempt is bound to fail. Exim remembers such failures in a domain cache record, which it uses to fail callouts for the domain without making new connections, until the domain record times out. There are two separate expiry times for domain cache records: **callout_domain_negative_expire** (default 3h) and **callout_domain_positive_expire** (default 7d).

Domain records expire when the negative expiry time is reached if callouts cannot be made for the domain, or if the postmaster check failed. Otherwise, they expire when the positive expiry time is reached. This ensures that, for example, a host that stops accepting 'random' local parts will eventually be noticed.

The callout caching mechanism is based entirely on the domain of the address that is being tested. If the domain routes to several hosts, it is assumed that their behaviour will be the same.

37.17 Sender address verification reporting

When sender verification fails in an ACL, the details of the failure are given as additional output lines before the 550 response to the relevant SMTP command (RCPT or DATA). For example, if sender callout is in use, you might see:

```
MAIL FROM:<xyz@abc.example>
250 OK
RCPT TO:<pqr@def.example>
550-Verification failed for <xyz@abc.example>
550-Called:   192.168.34.43
550-Sent:     RCPT TO:<xyz@abc.example>
550-Response: 550 Unknown local part xyz in <xyz@abc.example>
550 Sender verification failed
```

If more than one RCPT command fails in the same way, the details are given only for the first of them. However, some administrators do not want to send out this much information. You can suppress the details by adding '/no_details' to the ACL statement that requests sender verification. For example:

```
verify = sender/no_details
```

37.18 Redirection while verifying

A dilemma arises when a local address is redirected by aliasing or forwarding during verification: should the generated addresses themselves be verified, or should the successful expansion of the original address be enough to verify it? Exim takes the following pragmatic approach:

- When an incoming address is redirected to just one child address, verification continues with the child address, and if that fails to verify, the original verification also fails.
- When an incoming address is redirected to more than one child address, verification does not continue. A success result is returned.

This seems the most reasonable behaviour for the common use of aliasing as a way of redirecting different local parts to the same mailbox. It means, for example, that a pair of alias entries of the form

```
A.Wol:   aw123
aw123:   :fail: Gone away, no forwarding address
```

work as expected, with both local parts causing verification failure. When a redirection generates more than one address, the behaviour is more like a mailing list, where the existence of the alias itself is sufficient for verification to succeed.

37.19 Using an ACL to control relaying

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. Redirecting a local address via an alias or forward file and then passing the message on to another host is not relaying, but a redirection as a result of the ‘percent hack’ is.

Two kinds of relaying exist, which are 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. On the other hand, 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. What is not wanted is the transmission of mail from arbitrary remote hosts through your system to arbitrary domains.

You can implement relay control by means of suitable statements in the ACL that runs for each RCPT command. For convenience, it is often easiest to use Exim’s named list facility to define the domains and hosts involved. For example, suppose you want to do the following:

- Deliver a number of domains to mailboxes on the local host (or process them locally in some other way). Let’s say these are *my.dom1.example* and *my.dom2.example*.
- Relay mail for a number of other domains for which you are the secondary MX. These might be *friend1.example* and *friend2.example*.
- Relay mail from the hosts on your local LAN, to whatever domains are involved. Suppose your LAN is 192.168.45.0/24.

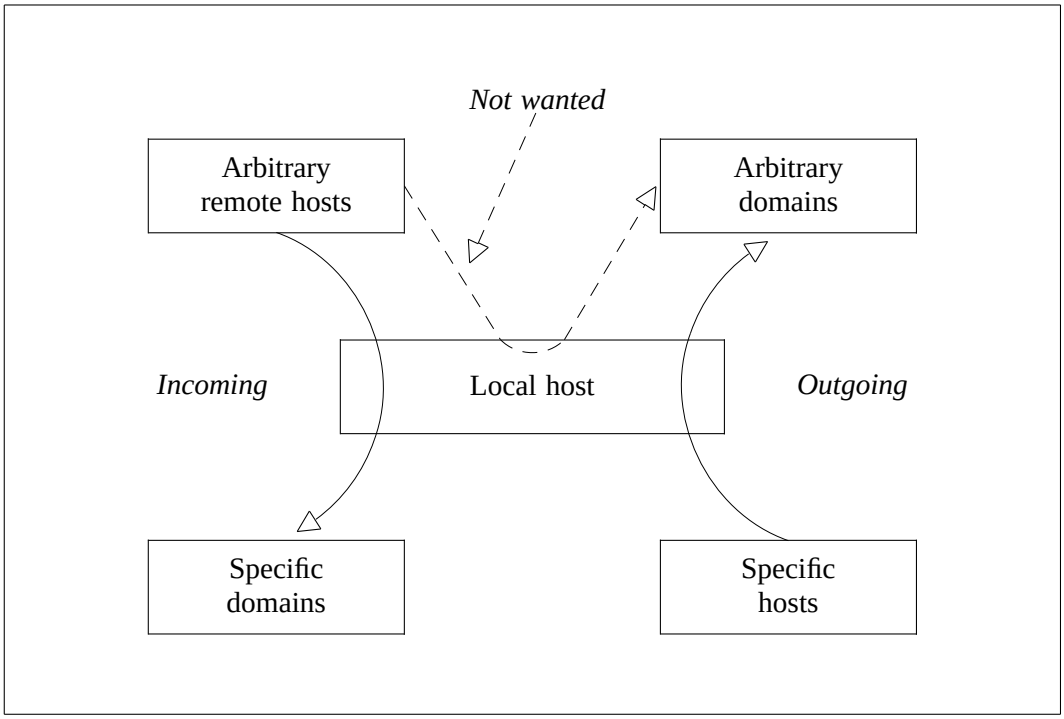
In the main part of the configuration, you put the following definitions:

```
domainlist local_domains = my.dom1.example : my.dom2.example
domainlist relay_domains = friend1.example : friend2.example
hostlist   relay_hosts   = 192.168.45.0/24
```

Now you can use these definitions in the ACL that is run for every RCPT command:

```
acl_check_rcpt:
  accept domains = +local_domains : +relay_domains
  accept hosts   = +relay_hosts
```

The first statement accepts any RCPT command that contains an address in the local or relay domains. For any other domain, control passes to the second statement, which accepts the command only if it comes from one of the relay hosts. In practice, you will probably want to make your ACL more



Controlled relaying

sophisticated than this, for example, by including sender and recipient verification. The default configuration includes a more comprehensive example, which is described in chapter 7.

37.20 Checking a relay configuration

You can check the relay characteristics of your configuration in the same way that you can test any ACL behaviour for an incoming SMTP connection, by using the **-bh** option to run a fake SMTP session with which you interact.

For specifically testing for unwanted relaying, the host *relay-test.mail-abuse.org* provides a useful service. If you telnet to this host from the host on which Exim is running, using the normal telnet port, you will see a normal telnet connection message and then quite a long delay. Be patient. The remote host is making an SMTP connection back to your host, and trying a number of common probes to test for open relay vulnerability. The results of the tests will eventually appear on your terminal.

38. Adding a local scan function to Exim

In these days of email worms, viruses, and ever-increasing spam, some sites want to apply a lot of checking to messages before accepting them. You can do a certain amount through string expansions and the **condition** condition in the ACL that runs after the SMTP DATA command or the ACL for non-SMTP messages (see chapter 37), but this has its limitations. To allow for more general checking that can be customized to a site's own requirements, there is the possibility of linking Exim with a private message scanning function, written in C. If you want to run code that is written in something other than C, you can of course use a little C stub to call it.

The local scan function is run for every incoming message. It can therefore be used to control non-SMTP messages from local processes as well as messages arriving via SMTP.

Exim applies a timeout to calls of the local scan function, and there is an option called **local_scan_timeout** for setting it. The default is 5 minutes. Zero means 'no timeout'. Exim also sets up signal handlers for SIGSEGV, SIGILL, SIGFPE, and SIGBUS before calling the local scan function, so that the most common types of crash are caught. If the timeout is exceeded or one of those signals is caught, the incoming message is rejected with a temporary error if it is an SMTP message. For a non-SMTP message, the message is dropped and Exim ends with a non-zero code. The incident is logged on the main and reject logs.

38.1 Building Exim to use a local scan function

To make use of the local scan function feature, you must tell Exim where your function is before building Exim, by setting LOCAL_SCAN_SOURCE in your **Local/Makefile**. A recommended place to put it is in the **Local** directory, so you might set

```
LOCAL_SCAN_SOURCE=Local/local_scan.c
```

for example. The function must be called *local_scan()*. It is called by Exim after it has received a message, when the success return code is about to be sent. This is after all the ACLs have been run. The return code from your function controls whether the message is actually accepted or not. There is a commented template function (that just accepts the message) in the file **src/local_scan.c**.

If you want to make use of Exim's run time configuration file to set options for your *local_scan()* function, you must also set

```
LOCAL_SCAN_HAS_OPTIONS=yes
```

in **Local/Makefile** (see section 38.3 below).

38.2 API for local_scan()

You must include this line near the start of your code:

```
#include "local_scan.h"
```

This header file defines a number of variables and other values, and the prototype for the function itself. Exim is coded to use unsigned char values almost exclusively, and one of the things this header defines is a shorthand for `unsigned char` called `uschar`. It also contains the following macro definitions, to simplify casting character strings and pointers to character strings:

```
#define CS   (char *)
#define CCS  (const char *)
#define CSS  (char **)
#define US   (unsigned char *)
#define CUS  (const unsigned char *)
#define USS  (unsigned char **)
```

The function prototype for *local_scan()* is:

```
extern int local_scan(int fd, uschar **return_text);
```

The arguments are as follows:

- **fd** is a file descriptor for the file that contains the body of the message (the -D file). The file is open for reading and writing, but updating it is not recommended. **Warning**: You must *not* close this file descriptor.

  The descriptor is positioned at character 19 of the file, which is the first character of the body itself, because the first 19 characters are the message id followed by `-D` and a newline. If you rewind the file, you should use the macro SPOOL_DATA_START_OFFSET to reset to the start of the data, just in case this changes in some future version.

- **return_text** is an address which you can use to return a pointer to a text string at the end of the function. The value it points to on entry is NULL.

The function must return an **int** value which is one of the following macros:

- `LOCAL_SCAN_ACCEPT`

  The message is accepted. If you pass back a string of text, it is saved with the message, and made available in the variable **$local_scan_data**. No newlines are permitted (if there are any, they are turned into spaces) and the maximum length of text is 1000 characters.

- `LOCAL_SCAN_ACCEPT_FREEZE`

  This behaves as LOCAL_SCAN_ACCEPT, except that the accepted message is queued without immediate delivery, and is frozen.

- `LOCAL_SCAN_ACCEPT_QUEUE`

  This behaves as LOCAL_SCAN_ACCEPT, except that the accepted message is queued without immediate delivery.

- `LOCAL_SCAN_REJECT`

  The message is rejected; the returned text is used as an error message which is passed back to the sender and which is also logged. Newlines are permitted – they cause a multiline response for SMTP rejections, but are converted to `\n` in log lines. If no message is given, 'Administrative prohibition' is used.

- `LOCAL_SCAN_TEMPREJECT`

  The message is temporarily rejected; the returned text is used as an error message as for LOCAL_SCAN_REJECT. If no message is given, 'Temporary local problem' is used.

- `LOCAL_SCAN_REJECT_NOLOGHDR`

  This behaves as LOCAL_SCAN_REJECT, except that the header of the rejected message is not written to the reject log. It has the effect of unsetting the **rejected_header** log selector for just this rejection. If **rejected_header** is already unset (see the discussion of the **log_selection** option in section 44.15), this code is the same as LOCAL_SCAN_REJECT.

- `LOCAL_SCAN_TEMPREJECT_NOLOGHDR`

  This code is a variation of LOCAL_SCAN_TEMPREJECT in the same way that LOCAL_SCAN_REJECT_NOLOGHDR is a variation of LOCAL_SCAN_REJECT.

If the message is not being received by interactive SMTP, rejections are reported by writing to **stderr** or by sending an email, as configured by the **-oe** command line options.

38.3 Configuration options for local_scan()

It is possible to have option settings in the main configuration file that set values in static variables in the *local_scan()* module. If you want to do this, you must have the line

```
LOCAL_SCAN_HAS_OPTIONS=yes
```

in your **Local/Makefile** when you build Exim. (This line is in **OS/Makefile-Default**, commented out). Then, in the *local_scan()* source file, you must define static variables to hold the option values, and a table to define them.

The table must be a vector called **local_scan_options**, of type `optionlist`. Each entry is a triplet, consisting of a name, an option type, and a pointer to the variable that holds the value. The entries must appear in alphabetical order. Following **local_scan_options** you must also define a variable called **local_scan_options_count** that contains the number of entries in the table. Here is a short example, showing two kinds of option:

```
static int my_integer_option = 42;
static uschar *my_string_option = US"a default string";

optionlist local_scan_options[] = {
  { "my_integer", opt_int,       &my_integer_option },
  { "my_string",  opt_stringptr, &my_string_option }
};
int local_scan_options_count =
  sizeof(local_scan_options)/sizeof(optionlist);
```

The values of the variables can now be changed from Exim's runtime configuration file by including a local scan section as in this example:

```
begin local_scan
my_integer = 99
my_string = some string of text...
```

The available types of option data are as follows:

opt_bool

This specifies a boolean (true/false) option. The address should point to a variable of type `BOOL`, which will be set to TRUE or FALSE, which are macros that are defined as '1' and '0', respectively. If you want to detect whether such a variable has been set at all, you can initialize it to TRUE_UNSET. (BOOL variables are integers underneath, so can hold more than two values.)

opt_fixed

This specifies a fixed point number, such as is used for load averages. The address should point to a variable of type `int`. The value is stored multiplied by 1000, so, for example, 1.4142 is truncated and stored as 1414.

opt_int

This specifies an integer; the address should point to a variable of type `int`. The value may be specified in any of the integer formats accepted by Exim.

opt_mkint

This is the same as **opt_int**, except that when such a value is output in a **-bP** listing, if it is an exact number of kilobytes or megabytes, it is printed with the suffix K or M.

opt_octint

This also specifies an integer, but the value is always interpeted as an octal integer, whether or not it starts with the digit zero, and it is always output in octal.

opt_stringptr

This specifies a string value; the address must be a pointer to a variable that points to a string (for example, of type `uschar *`).

opt_time

This specifies a time interval value. The address must point to a variable of type `int`. The value that is placed there is a number of seconds.

If the **-bP** command line option is followed by `local_scan`, Exim prints out the values of all the *local_scan()* options.

38.4 Available Exim variables

The header **local_scan.h** gives you access to a number of C variables. These are the only ones that are guaranteed to be maintained from release to release. Note, however, that you can obtain the value of any Exim variable by calling *expand_string()*. The exported variables are as follows:

unsigned int debug_selector

This variable is set to zero when no debugging is taking place. Otherwise, it is a bitmap of debugging selectors. Two bits are identified for use in *local_scan()*; they are defined as macros:

- The `D_v` bit is set when **-v** was present on the command line. This is a testing option that is not privileged – any caller may set it. All the other selector bits can be set only by admin users.
- The `D_local_scan` bit is provided for use by *local_scan()*; it is set by the `+local_scan` debug selector. It is not included in the default set of debugging bits.

Thus, to write to the debugging output only when `+local_scan` has been selected, you should use code like this:

```
if ((debug_selector & D_local_scan) != 0)
  debug_printf("xxx", ...);
```

uschar *expand_string_message

After a failing call to *expand_string()* (returned value NULL), the variable **expand_string_message** contains the error message, zero-terminated.

header_line *header_list

A pointer to a chain of header lines. The **header_line** structure is discussed below.

header_line *header_last

A pointer to the last of the header lines.

BOOL host_checking

This variable is TRUE during a host checking session that is initiated by the **-bh** command line option.

uschar *interface_address

The IP address of the interface that received the message, as a string. This is NULL for locally submitted messages.

int interface_port

The port on which this message was received.

uschar *message_id

This variable contains the message id for the incoming message as a zero-terminated string.

uschar *received_protocol

The name of the protocol by which the message was received.

int recipients_count

The number of accepted recipients.

recipient_item *recipients_list

The list of accepted recipients, held in a vector of length **recipients_count**. The **recipient_item** structure is discussed below. You can add additional recipients by calling *receive_add_recipient()* (see below). You can delete recipients by removing them from the vector and adusting the value in **recipients_count**. In particular, by setting **recipients_count** to zero you remove all recipients. If you then return the value `LOCAL_SCAN_ACCEPT`, the message is accepted, but immediately blackholed. To replace the recipients, set **recipients_count** to zero and then call *receive_add_recipient()* as often as needed.

uschar *sender_address

The envelope sender address. For bounce messages this is the empty string.

uschar *sender_host_address

The IP address of the sending host, as a string. This is NULL for locally-submitted messages.

uschar *sender_host_authenticated

The name of the authentication mechanism that was used, or NULL if the message was not received over an authenticated SMTP connection.

uschar *sender_host_name

The name of the sending host, if known.

int sender_host_port

The port on the sending host.

int store_pool

The contents of this variable control which pool of memory is used for new requests. See section 38.8 for details.

38.5 Structure of header lines

The **header_line** structure contains the members listed below. You can add additional header lines by calling the *header_add()* function (see below). You can cause header lines to be ignored (deleted) by setting their type to *.

struct header_line *next

A pointer to the next header line, or NULL for the last line.

int type

A code identifying certain headers that Exim recognizes. The codes are printing characters, and are documented in chapter 48 of this manual. Notice in particular that any header line whose type is * is not transmitted with the message. This flagging is used for header lines that have been rewritten, or are to be removed (for example, *Envelope-sender:* header lines.) Effectively, * means ‘deleted’.

int slen

The number of characters in the header line, including the terminating and any internal newlines.

uschar *text

A pointer to the text of the header. It always ends with a newline, followed by a zero byte. Internal newlines are preserved.

38.6 Structure of recipient items

The **recipient_item** structure contains these members:

uschar *address

This is a pointer to the recipient address as it was received.

int pno

This is used in later Exim processing when top level addresses are created by the **one_time** option. It is not relevant at the time *local_scan()* is run and must always contain -1 at this stage.

uschar *errors_to

If this value is not NULL, bounce messages caused by failing to deliver to the recipient are sent to the address it contains. In other words, it overrides the envelope sender for this one recipient. (Compare the **errors_to** generic router option.) If a *local_scan()* function sets an **errors_to** field to an unqualified address, Exim qualifies it using the domain from **qualify_recipient**. When *local_scan()* is called, the **errors_to** field is NULL for all recipients.

38.7 Available Exim functions

The header **local_scan.h** gives you access to a number of Exim functions. These are the only ones that are guaranteed to be maintained from release to release:

pid_t child_open(uschar **argv, uschar **envp, int newumask, int *infdptr, int *outfdptr, BOOL make_leader)

This function creates a child process that runs the command specified by **argv**. The environment for the process is specified by **envp**, which can be NULL if no environment variables are to be passed. A new umask is supplied for the process in **newumask**.

Pipes to the standard input and output of the new process are set up and returned to the caller via the **infdptr** and **outfdptr** arguments. The standard error is cloned to the standard output. If there are any file descriptors 'in the way' in the new process, they are closed. If the final argument is TRUE, the new process is made into a process group leader.

The function returns the pid of the new process, or -1 if things go wrong.

int child_close(pid_t pid, int timeout)

This function waits for a child process to terminate, or for a timeout (in seconds) to expire. A timeout value of zero means wait as long as it takes. The return value is as follows:

- >= 0

  The process terminated by a normal exit and the value is the process ending status.

- < 0 and > –256

  The process was terminated by a signal and the value is the negation of the signal number.

- –256

  The process timed out.

- –257

  The was some other error in wait(); **errno** is still set.

pid_t child_open_exim(int *fd)

This function provide you with a means of submitting a new message to Exim. (Of course, you can also call **/usr/sbin/sendmail** yourself if you want, but this packages it all up for you.) The function creates a pipe, forks a subprocess that is running

```
exim -t -oem -oi -f <>
```

and returns to you (via the `int *` argument) a file descriptor for the pipe that is connected to the standard input. The yield of the function is the PID of the subprocess. You can then write a message to the file descriptor, with recipients in *To:*, *Cc:*, and/or *Bcc:* header lines.

When you have finished, call *child_close()* to wait for the process to finish and to collect its ending status. A timeout value of zero is usually fine in this circumstance. Unless you have made a mistake with the recipient addresses, you should get a return code of zero.

void debug_printf(char *, ...)

This is Exim's debugging function, with arguments as for *(printf()*. The output is written to the standard error stream. If no debugging is selected, calls to *debug_printf()* have no effect. Normally, you should make calls conditional on the `local_scan` debug selector by coding like this:

```
if ((debug_selector & D_local_scan) != 0)
  debug_printf("xxx", ...);
```

uschar *expand_string(uschar *string)

This is an interface to Exim's string expansion code. The return value is the expanded string, or NULL if there was an expansion failure. The C variable **expand_string_message** contains an error message after an expansion failure. If expansion does not change the string, the return value is the pointer to the input string. Otherwise, the return value points to a new block of memory that was obtained by a call to *store_get()*. See section 38.8 below for a discussion of memory handling.

void header_add(int type, char *format, ...)

This function allows you to add additional header lines. The first argument is the type, and should normally be a space character. The second argument is a format string and any number of substitution arguments as for *sprintf()*. You may include internal newlines if you want, and you must ensure that the string ends with a newline.

uschar *lss_b64encode(uschar *cleartext, int length)

This function base64-encodes a string, which is passed by address and length. The text may contain bytes of any value, including zero. The result is passed back in dynamic memory that is obtained by calling *store_get()*. It is zero-terminated.

int lss_b64decode(uschar *codetext, uschar **cleartext)

This function decodes a base64-encoded string. Its arguments are a zero-terminated base64-encoded string and the address of a variable that is set to point to the result, which is in dynamic memory. The length of the decoded string is the yield of the function. If the input is invalid base64 data, the yield is -1. A zero byte is added to the end of the output string to make it easy to interpret as a C string (assuming it contains no zeros of its own). The added zero byte is not included in the returned count.

int lss_match_domain(uschar *domain, uschar *list)

This function checks for a match in a domain list. Domains are always matched caselessly. The return value is one of the following:

`OK`	match succeeded
`FAIL`	match failed
`DEFER`	match deferred

DEFER is usually caused by some kind of lookup defer, such as the inability to contact a database.

int lss_match_local_part(uschar *localpart, uschar *list, BOOL caseless)

This function checks for a match in a local part list. The third argument controls case-sensitivity. The return values are as for *lss_match_domain()*.

int lss_match_address(uschar *address, uschar *list, BOOL caseless)

This function checks for a match in an address list. The third argument controls the case-sensitivity of the local part match. The domain is always matched caselessly. The return values are as for *lss_match_domain()*.

int lss_match_host(uschar *host_name, uschar *host_address, uschar *list)

This function checks for a match in a host list. The most common usage is expected to be

```
lss_match_host(sender_host_name, sender_host_address, ...)
```

An empty address field matches an empty item in the host list. If the host name is NULL, the name corresponding to **$sender_host_address** is automatically looked up if a host name is required to match an item in the list. The return values are as for *lss_match_domain()*, but in addition, *lss_match_host()* returns ERROR in the case when it had to look up a host name, but the lookup failed.

void log_write(unsigned int selector, int which, char *format, ...)

This function writes to Exim's log files. The first argument should be zero (it is concerned with **log_selector**). The second argument can be `LOG_MAIN` or `LOG_REJECT` or `LOG_PANIC` or the inclusive 'or' of any combination of them. It specifies to which log or logs the message is written. The remaining arguments are a format and relevant insertion arguments. The string should not contain any newlines, not even at the end.

void receive_add_recipient(uschar *address, int pno)

This function adds an additional recipient to the message. The first argument is the recipient address. If if is unqualified (has no domain), it is qualified with the **qualify_recipient** domain. The second argument must always be -1.

This function does not allow you to specify a private **errors_to** address (as described with the structure of **recipient_item** above), because it pre-dates the addition of that field to the structure. However, it is easy to add such a value afterwards. For example:

```
receive_add_recipient(US"monitor@mydom.example", -1);
recipients_list[recipients_count-1].errors_to =
  US"postmaster@mydom.example";
```

void *store_get(int)

This function accesses Exim's internal store (memory) manager. It gets a new chunk of memory whose size is given by the argument. Exim bombs out if it ever runs out of memory. See the next section for a discussion of memory handling.

void *store_get_perm(int)

This function is like *store_get()*, but it always gets memory from the permanent pool. See the next section for a discussion of memory handling.

uschar *string_copy(uschar *string)

uschar *string_copyn(uschar *string, int length)

uschar *string_sprintf(char *format, ...)

These three functions create strings using Exim's dynamic memory facilities. The first makes a copy of an entire string. The second copies up to a maximum number of characters, indicated by

the second argument. The third uses a format and insertion arguments to create a new string. In each case, the result is a pointer to a new string in the current memory pool. See the next section for more discussion.

38.8 More about Exim's memory handling

No function is provided for freeing memory, because that is never needed. The dynamic memory that Exim uses when receiving a message is automatically recycled if another message is received by the same process (this applies only to incoming SMTP connections – other input methods can supply only one message at a time). After receiving the last message, a reception process terminates.

Because it is recycled, the normal dynamic memory cannot be used for holding data that must be preserved over a number of incoming messages on the same SMTP connection. However, Exim in fact uses two pools of dynamic memory; the second one is not recycled, and can be used for this purpose.

If you want to allocate memory that remains available for subsequent messages in the same SMTP connection, you should set

```
store_pool = POOL_PERM
```

before calling the function that does the allocation. There is no need to restore the value if you do not need to; however, if you do want to revert to the normal pool, you can either restore the previous value of **store_pool** or set it explicitly to POOL_MAIN.

The pool setting applies to all functions that get dynamic memory, including *expand_string()*, *store_get()*, and the *string_xxx()* functions. There is also a convenience function called *store_get_perm()* that gets a block of memory from the permanent pool while preserving the value of **store_pool**.

39. System-wide message filtering

The previous chapters (on ACLs and the local scan function) describe checks that can be applied to messages before they are accepted by a host. There is also a mechanism for checking messages once they have been received, but before they are delivered. This is called the *system filter*.

The system filter operates in a similar manner to users' filter files, but it is run just once per message (however many recipients the message has). It should not normally be used as a substitute for routing, because **deliver** commands in a system router provide new envelope recipient addresses.

The system filter is run at the start of a delivery attempt, before any routing is done. If a message fails to be completely delivered at the first attempt, the system filter is run again at the start of every retry. If you want your filter to do something only once per message, you can make use of the **first_delivery** condition in an **if** command in the filter to prevent it happening on retries.

Warning: Because the system filter runs just once, variables that are specific to individual recipient addresses, such as **$local_part** and **$domain**, are not set, and the 'personal' condition is not meaningful. If you want to run a centrally-specified filter for each recipient address independently, you can do so by setting up a suitable **redirect** router, as described in section 39.8 below.

39.1 Specifying a system filter

The name of the file that contains the system filter must be specified by setting **system_filter**. If you want the filter to run under a uid and gid other than root, you must also set **system_filter_user** and **system_filter_group** as appropriate. For example:

```
system_filter = /etc/mail/exim.filter
system_filter_user = exim
```

If a system filter generates any deliveries directly to files or pipes (via the **save** or **pipe** commands), transports to handle these deliveries must be specified by setting **system_filter_file_transport** and **system_filter_pipe_transport**, respectively. Similarly, **system_filter_reply_transport** must be set to handle any messages generated by the **reply** command.

39.2 Testing a system filter

You can run simple tests of a system filter in the same way as for a user filter, but you should use **-bF** rather than **-bf**, so that features that are permitted only in system filters are recognized.

39.3 Contents of a system filter

The language used to specify system filters is the same as for users' filter files. It is described in the separate end-user document *Exim's interface to mail filtering*. However, there are some additional features that are available only in system filters; these are described in subsequent sections. If they are encountered in a user's filter file or when testing with **-bf**, they cause errors.

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, and **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.

Warning: If a system filter uses the **first_delivery** condition to specify an 'unseen' (non-significant) delivery, and that delivery does not succeed, it will not be tried again. If you want Exim to retry an unseen delivery until it succeeds, you should arrange to set it up every time the filter runs.

When a system filter finishes running, the values of the variables **$n0** – **$n9** are copied into **$sn0** – **$sn9** and are thereby made available to users' filter files. Thus a system filter can, for example, set up 'scores' to which users' filter files can refer.

39.4 Additional variable for system filters

The expansion variable **$recipients**, containing a list of all the recipients of the message (separated by commas and white space), is available in system filters. It is not available in users' filters for privacy reasons.

39.5 Defer, freeze, and fail commands for system filters

There are three extra commands (**defer**, **freeze** and **fail**) which are always available in system filters, but are not normally enabled in users' filters. (See the **allow_defer**, **allow_freeze** and **allow_fail** options for the **redirect** router.) These 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"
```

The keyword **text** is optional if the next character is a double quote.

The **defer** command defers delivery of the original recipients of the message. The **fail** command causes all the original recipients to be failed, and a bounce message to be created. The **freeze** command suspends all delivery attempts for the original recipients. In all cases, any new deliveries that are specified by the filter are attempted as normal after the filter has run.

The **freeze** command 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 text given with a fail command is used as part of the bounce message as well as being written to the log. If the message is quite long, this can fill up a lot of log space when such failures are common. To reduce the size of the log message, Exim interprets the text in a special way if it starts with the two characters `<<` and contains `>>` later. The text between these two strings is written to the log, and the rest of the text is used in the bounce message. For example:

```
fail "<<filter test 1>>Your message is rejected \
     because it contains attachments that we are \
     not prepared to receive."
```

Take great care with the **fail** command when basing the decision to fail on the contents of the message, because the bounce message 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 let through unwanted bounce messages. The alternative is clever checking of the body and/or headers to detect bounces generated by the filter.

The interpretation of a system filter file ceases after a **defer**, **freeze**, or **fail** command is obeyed. However, any deliveries that were set up earlier in the filter file are honoured, so you can use a sequence such as

```
mail ...
freeze
```

to send a specified message when the system filter is freezing (or deferring or failing) a message. The normal deliveries for the message do not, of course, take place.

39.6 Adding and removing headers in a system filter

Two filter commands that are available only in system filters are:

```
headers add <<string>>
headers remove <<string>>
```

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 2822 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.

If the message is not delivered at the first attempt, header lines that were added by the system filter are stored with the message, and so are still present at the next delivery attempt. For that reason, it is usual to make the **headers add** command conditional on **first_delivery**.

You can use '\n' within the string, followed by white space, to specify continued header lines. More than one header may be added in one command by including '\n' within the string without any following white space (though this is not recommended, for a reason given below). For example:

```
headers add "X-header-1: ....\n  \
             continuation of X-header-1 ...\n\
             X-header-2: ...."
```

Note that the header line continuation white space after the first newline must be placed before the backslash that continues the input string, because white space after input continuations is ignored.

Header lines that are added by a system filter are visible to users' filter files and to all routers and transports. However, if you add multiple header lines in a single **headers add** command, the entire text is returned if a subsequent string expansion refers to the first of them; the second and subsequent header lines are not accessible. If you want the individual added header lines to be accessible in string expansions, you must use a separate **headers add** command for each one.

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; those that are added at delivery time (such as *Envelope-To:* and *Return-Path:*) cannot be removed by this means. If there is more than one header with the same name, they are all removed.

39.7 Setting an errors address in a system filter

In a system filter, if a **deliver** command is followed by

```
errors_to <some address>
```

in order to change the envelope sender (and hence the error reporting) for that delivery, any address may be specified. (In a user filter, only the current user's address can be set.) For example, if some mail is being monitored, you might use

```
unseen deliver monitor@spying.example errors_to root@local.example
```

to take a copy which would not be sent back to the normal error reporting address if its delivery failed.

39.8 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 recipient address. 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 router which implements such a filter:

```
central_filter:
  driver = redirect
  domains = +local_domains
  file = /central/filters/$local_part
  no_verify
  allow_filter
  allow_freeze
```

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 router will not then

claim to have dealt with the address, so it will be passed on to subsequent routers to be delivered in the normal way.

40. Customizing bounce and warning messages

When a message fails to be 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.

40.1 Customizing bounce messages

If **bounce_message_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 **bounce_message_file** is set.

When **bounce_message_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: **$bounce_recipient** is set to the recipient of an error message while it is being created, and **$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 (**bounce_message_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}{$bounce_recipient}{: returning message to sender}}
****
This message was created automatically by mail delivery software.

A message ${if eq{$sender_address}{$bounce_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.
****
```

40.2 Customizing warning messages

The option **warn_message_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 following file, except that the line starting 'A message' has been split here, in order to fit it on the page:

```
Subject: Warning: message $message_id delayed $warn_message_delay
****
This message was created automatically by mail delivery software.

A message ${if eq{$sender_address}{$warn_message_recipients}
  {that you sent }{sent by

  <$sender_address>

}}has not been delivered to all of its recipients after
more than $warn_message_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, **$warn_message_delay** is set to the delay time in one of the forms '<*n*> minutes' or '<*n*> hours', and **$warn_message_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 that handled them.

41. Some common configuration requirements

This chapter discusses some configuration requirements that seem to be fairly common. More examples and discussion can be found in the Exim book.

41.1 Sending mail to a smart host

If you want to send all mail for non-local domains to a ‘smart host’, you should replace the default **dnslookup** router with a router which does the routing explicitly:

```
send_to_smart_host:
  driver = manualroute
  route_list = !+local_domains smart.host.name
  transport = remote_smtp
```

You can use the smart host’s IP address instead of the name if you wish.

41.2 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 such as Majordomo or Mailman is recommended.

The **redirect** router 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** router option can be used to run these lists in a separate domain from normal mail. For example:

```
lists:
  driver = redirect
  domains = lists.example
  file = /usr/lists/$local_part
  forbid_pipe
  forbid_file
  errors_to = $local_part-request@lists.example
  no_more
```

This router is skipped for domains other than *lists.example*. For addresses in that domain, it looks for a file that matches the local part. If there is no such file, the router declines, but because **no_more** is set, no subsequent routers are tried, and so the whole delivery fails.

The **forbid_pipe** and **forbid_file** options prevent a local part from being expanded into a file name or a pipe delivery, which is usually inappropriate in a mailing list.

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.example* is passed on to those addresses contained in **/usr/lists/dicts**, with error reports directed to *dicts-request@lists.example*, provided that this address can be verified. There could be a file called **/usr/lists/dicts-request** containing the address(es) of this particular list’s manager(s), but other approaches, such as setting up an earlier router (possibly using the **local_part_prefix** or **local_part_suffix** options) to handle addresses of the form **owner-xxx** or **xxx-request**, are also possible.

41.3 Syntax errors in mailing lists

If an entry in redirection data contains a syntax error, Exim normally defers delivery of the original address. That means that a syntax error in a mailing list holds up all deliveries to the list. This may not be appropriate when a list is being maintained automatically from data supplied by users, and the addresses are not rigorously checked.

If the **skip_syntax_errors** option is set, the **redirect** router just skips entries that fail to parse, noting the incident in the log. If in addition **syntax_errors_to** is set to a verifiable address, a message is sent to it whenever a broken address is skipped. It is usually appropriate to set **syntax_errors_to** to the same address as **errors_to**.

41.4 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, so addresses that have been added to the list since the message arrived will therefore 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 **redirect** router. If this is done, any addresses generated by the router 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 the **all_parents** selector is set, but for mailing lists there is normally only one level of expansion anyway.

41.5 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 to restrict the router that handles the list.

The following example uses the same file as a list of recipients and as a list of permitted senders. It requires three routers:

```
lists_request:
  driver = redirect
  domains = lists.example
  local_part_suffix = -request
  file = /usr/lists/$local_part$local_part_suffix
  no_more

lists_post:
  driver = redirect
  domains = lists.example
  senders = ${if exists {/usr/lists/$local_part}\
             {lsearch;/usr/lists/$local_part}{*}}
  file = /usr/lists/$local_part
  forbid_pipe
  forbid_file
  errors_to = $local_part-request@lists.example
  no_more

lists_closed:
  driver = redirect
  domains = lists.example
  allow_fail
  data = :fail: $local_part@lists.example is a closed mailing list
```

All three routers have the same **domains** setting, so for any other domains, they are all skipped. The first router runs only if the local part ends in **-request**. It handles messages to the list manager(s) by means of an open mailing list.

The second router runs only if the **senders** precondition is satisfied. It checks for the existence of a list that corresponds to the local part, and then checks that the sender is on the list by means of a linear search. It is necessary to check for the existence of the file before trying to search it, because otherwise Exim thinks there is a configuration error. If the file does not exist, the expansion of **senders** is *, which matches all senders. This means that the router runs, but because there is no list, declines, and **no_more** ensures that no further routers are run. The address fails with an 'unrouteable address' error.

The third router runs only if the second router is skipped, which happens when a mailing list exists, but the sender is not on it. This router forcibly fails the address, giving a suitable error message.

41.6 Virtual domains

The phrase *virtual domain* is unfortunately used with two rather different meanings:

- A domain for which there are no real mailboxes; all valid local parts are aliases for other email addresses. Common examples are organizational top-level domains and 'vanity' domains.
- One of a number of independent domains that are all handled by the same host, with mailboxes on that host, but where the mailbox owners do not necessarily have login accounts on that host.

The first usage is probably more common, and does seem more 'virtual' than the second. This kind of domain can be handled in Exim with a straightforward aliasing router. One approach is to create a separate alias file for each virtual domain. Exim can test for the existence of the alias file to determine whether the domain exists. The **dsearch** lookup type is useful here, leading to a router of this form:

```
virtual:
  driver = redirect
  domains = dsearch;/etc/mail/virtual
  data = ${lookup{$local_part}lsearch{/etc/mail/virtual/$domain}}
  no_more
```

The **domains** option specifies that the router is to be skipped, unless there is a file in the **/etc/mail/virtual** directory whose name is the same as the domain that is being processed. When the router runs, it looks up the local part in the file to find a new address (or list of addresses). The **no_more** setting ensures that if the lookup fails (leading to **data** being an empty string), Exim gives up on the address without trying any subsequent routers.

This one router can handle all the virtual domains because the alias file names follow a fixed pattern. Permissions can be arranged so that appropriate people can edit the different alias files. A successful aliasing operation results in a new envelope recipient address, which is then routed from scratch.

The other kind of 'virtual' domain can also be handled in a straightforward way. One approach is to create a file for each domain containing a list of valid local parts, and use it in a router like this:

```
my_domains:
  driver = accept
  domains = dsearch;/etc/mail/domains
  local_parts = lsearch;/etc/mail/domains/$domain
  transport = my_mailboxes
```

The address is accepted if there is a file for the domain, and the local part can be found in the file. The **domains** option is used to check for the file's existence because **domains** is tested before the **local_parts** option (see section 3.9). You can't use **require_files**, because that option is tested after **local_parts**. The transport is as follows:

```
my_mailboxes:
  driver = appendfile
  file = /var/mail/$domain/$local_part
  user = mail
```

This uses a directory of mailboxes for each domain. The **user** setting is required, to specify which uid is to be used for writing to the mailboxes.

The configuration shown here is just one example of how you might support this requirement. There are many other ways this kind of configuration can be set up, for example, by using a database instead of separate files to hold all the information about the domains.

41.7 Multiple user mailboxes

Heavy email users often want to operate with multiple mailboxes, into which incoming mail is automatically sorted. A popular way of handling this is to allow users to use multiple sender addresses, so that replies can easily be identified. Users are permitted to add prefixes or suffixes to their local parts for this purpose. The wildcard facility of the generic router options **local_part_prefix** and **local_part_suffix** can be used for this. For example, consider this router:

```
userforward:
  driver = redirect
  check_local_user
  file = $home/.forward
  local_part_suffix = -*
  local_part_suffix_optional
  allow_filter
```

It 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 routers, and, assuming no subsequent use of the **local_part_suffix** option is made, they presumably fail. Thus, users have control over which 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 = redirect
  check_local_user
  file = $home/.forward$local_part_suffix
  local_part_suffix = -*
  local_part_suffix_optional
  allow_filter
```

If there is no suffix, **.forward** is used; if the suffix is *-special*, for example, **.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 routers, which could, if required, look for an unqualified **.forward** file to use as a default.

41.8 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 router which can cause the message to be delivered directly to the *vacation* program, or alternatively can use 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 router 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.

41.9 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.

41.10 Intermittently connected hosts

It has become 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.

41.11 Exim on the upstream server host

It is tempting to arrange for incoming mail for the intermittently connected host to remain on Exim’s queue until the client connects. However, this approach 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 delivering these messages into local files in batch SMTP, ‘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.

On a very small scale, leaving the mail on Exim’s queue can be made to work. If you are doing this, you should configure Exim with a long retry period for the intermittent host. For example:

```
cheshire.wonderland.fict.example     *   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 section 42.9) 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, 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.

41.12 Exim on the intermittently connected client 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, because 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.

42. SMTP processing

Exim supports a number of different ways of using the SMTP protocol, and its LMTP variant, which is an interactive protocol for transferring messages into a closed mail store application. This chapter contains details of how SMTP is processed. For incoming mail, the following are available:

- SMTP over TCP/IP (Exim daemon or *inetd*);
- SMTP over the standard input and output (the **-bs** option);
- Batched SMTP on the standard input (the **-bS** option).

For mail delivery, the following are available:

- SMTP over TCP/IP (the **smtp** transport);
- LMTP over TCP/IP (the **smtp** transport with the **protocol** option set to 'lmtp');
- LMTP over a pipe to a process running in the local host (the **lmtp** transport);
- Batched SMTP to a file or pipe (the **appendfile** and **pipe** transports with the **use_bsmtp** option set).

Batched SMTP is the name for a process in which batches of messages are stored in or read from files (or pipes), in a format in which SMTP commands are used to contain the envelope information.

42.1 Outgoing SMTP and LMTP over TCP/IP

Outgoing SMTP and LMTP over TCP/IP is implemented by the **smtp** transport. The **protocol** option selects which protocol is to be used, but the actual processing is the same in both cases.

If, in response to its EHLO command, Exim 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.

If the remote server advertises support for PIPELINING, Exim uses the pipelining extension to SMTP (RFC 2197) to reduce the number of TCP/IP packets required for the transaction.

If the remote server advertises support for the STARTTLS command, and Exim was built to support TLS encryption, it tries to start a TLS session unless the server matches **hosts_avoid_tls**. See chapter 36 for more details.

If the remote server advertises support for the AUTH command, Exim scans the authenticators configuration for any suitable client settings, as described in chapter 32.

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 so that it can deliver another message using the same socket. 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 **connection_max_messages** option of the **smtp** transport can be used to limit the number of messages sent down a single TCP/IP connection. The second and subsequent messages delivered down an existing connection are identified in the main log by the addition of an asterisk after the closing square bracket of the IP address.

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 EHLO or HELO,
- 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.

For a host error, a permanent error response on connection, or in response to EHLO, causes all addresses routed to the host to be failed. Any other 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.

For a message error, 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, Exim adds SIZE=*nnn* to the MAIL command, so an over-large message will cause a message error because the error arrives 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.

For a recipient error, a permanent error response (5*xx*) causes the recipient address to be failed, and a bounce message 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. 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, it is 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 do not 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.dom.example\$}\
  {$1-request=$local_part%$domain@your.dom.example}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.dom.example*. The rewriting inserts the local part and domain of the recipient into the return path. Suppose, for example, that a message whose return path has been set to *somelist-request@your.dom.example* is sent to *subscriber@other.dom.example*. In the transport, the return path is rewritten as

```
somelist-request=subscriber%other.dom.example@your.dom.example
```

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 an **local_part_suffix** option for a suitable router.

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 locally running user agent using the **-bs** option by checking whether or not the standard input is a socket. When it is, either the port must be privileged (less than 1024), or the caller must be root or the Exim user. If any other user passes a socket with an unprivileged port number, Exim prints a message on the standard error stream and exits with an error code.

By default, Exim does not make a log entry when a remote host 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 **smtp_connection** log selector.

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.

One area that sometimes gives rise to problems concerns the EHLO or HELO commands. Some clients send syntactically invalid versions of these commands, which Exim rejects by default. (This is nothing to do with verifying the data that is sent, so **helo_verify_hosts** is not relevant.) You can tell Exim not to apply a syntax check by setting **helo_accept_junk_hosts** to match the broken hosts that send invalid commands.

The amount of disk 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 is not enough space. 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.

The Exim daemon does not rely on the SIGCHLD signal to detect when a subprocess has finished, as this can get lost at busy times. Instead, it looks for completed subprocesses every time it wakes up. Provided there are other things happening (new incoming calls, starts of queue runs), completed processes will be noticed and tidied away. On very quiet systems you may sometimes see a 'defunct' Exim process hanging about. This is not a problem; it will be noticed when the daemon next wakes up.

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.

Exim normally starts a delivery process for each message received, though this can be varied by means of the **-odq** command line option and the **queue_only**, **queue_only_file**, and **queue_only_load** options. The number of simultaneously running delivery processes started in this way from SMTP input can be limited by the **smtp_accept_queue** and **smtp_accept_queue_per_connection** options. When either limit is reached, subsequently received messages are just put on the input queue without starting a delivery process.

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, because in that case 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 37 for details. It can also be configured to rewrite addresses at this time – before any syntax checking is done. See section 30.7.

Exim can also be configured to limit the rate at which a client host submits MAIL and RCPT commands in a single SMTP session. See the **smtp_ratelimit_hosts** option.

42.5 Unrecognized SMTP commands

If Exim receives more than **smtp_max_unknown_commands** unrecognized SMTP commands during a single SMTP connection, it drops the connection after sending the error response to the last command. The default value for **smtp_max_unknown_commands** is 3. This is a defence against some kinds of abuse that subvert web servers into making connections to SMTP ports; in these circumstances, a number of non-SMTP lines are sent first.

42.6 Use of non-mail SMTP commands

42.7 SMTP||non-mail commands

The 'non-mail' SMTP commands are those other than MAIL, RCPT, and DATA. Exim counts such commands, and drops the connection if there are too many of them in a single SMTP session. This action catches some denial-of-service attempts and things like repeated failing AUTHs, or a mad client looping sending EHLO. The global option **smtp_accept_max_nonmail** defines what 'too many' means. Its default value is 10.

When a new message is expected, one occurrence of RSET is not counted. This allows a client to send one RSET between messages (this is not necessary, but some clients do it). Exim also allows one

uncounted occurence of HELO or EHLO, and one occurrence of STARTTLS between messages. After starting up a TLS session, another EHLO is expected, and so it too is not counted.

The first occurrence of AUTH in a connection, or immediately following STARTTLS is also not counted. Otherwise, all commands other than MAIL, RCPT, DATA, and QUIT are counted.

You can control which hosts are subject to the limit set by **smtp_accept_max_nonmail** by setting **smtp_accept_max_nonmail_hosts**. The default value is `*`, which makes the limit apply to all hosts. This option means that you can exclude any specific badly-behaved hosts that you have to live with.

42.8 The VRFY and EXPN commands

When Exim receives a VRFY or EXPN command on a TCP/IP connection, it runs the ACL specified by **acl_smtp_vrfy** or **acl_smtp_expn** (as appropriate) in order to decide whether the command should be accepted or not. If no ACL is defined, the command is rejected.

When VRFY is accepted, it runs exactly the same code as when Exim is called with the **-bv** option. When EXPN is accepted, 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.

42.9 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). When Exim receives an ETRN command on a TCP/IP connection, it runs the ACL specified by **acl_smtp_etrn** in order to decide whether the command should be accepted or not. If no ACL is defined, the command is rejected.

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, 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
```

runs the command

```
exim -R brigadoon
```

which causes a delivery attempt on all messages with undelivered addresses containing the text 'brigadoon'. When **smtp_etrn_serialize** is set (the default), Exim prevents the simultaneous execution of more than one queue run for the same argument string as a result of an ETRN command. This stops a misbehaving client from starting more than one queue runner at once.

Exim implements the serialization by means of a hints database in which a record is written whenever a process is started by ETRN, and deleted when the process completes. However, Exim does not keep the SMTP session waiting for the ETRN process to complete. Once ETRN is accepted, the client is sent a 'success' return code. 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.

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, whatever the form of its argument. 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. Exim does not wait for the command to complete, so its status code is not checked.

Exim runs under its own uid and gid when receiving incoming SMTP, so it is not possible for it to change them before running the command.

42.10 Incoming local SMTP

Some user agents use SMTP to pass messages to their local MTA using the standard input and output, as opposed to passing the envelope on the command line and writing the message to the standard input. This is supported by the **-bs** option. This form of SMTP is handled in the same way as incoming messages over TCP/IP (including the use of ACLs), except that the envelope sender given in a MAIL command is ignored unless the caller is trusted. In an ACL you can detect this form of SMTP input by testing for an empty host identification. It is common to have this as the first line in the ACL that runs for RCPT commands:

```
accept hosts = :
```

This accepts SMTP messages from local processes without doing any other tests.

42.11 Outgoing batched SMTP

Both the **appendfile** and **pipe** transports can be used for handling batched SMTP. Each has an option called **use_bsmtp** which causes messages to be output in BSMTP format. No SMTP responses are possible for this form of delivery. All it is doing is using SMTP commands as a way of transmitting the envelope along with the message.

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. Lines in the message that start with a dot have an extra dot added. The SMTP command HELO is not normally used. If it is required, the **message_prefix** option can be used to specify it.

Because **appendfile** and **pipe** are both local transports, they accept only one recipient address at a time by default. However, you can arrange for them to handle several addresses at once by setting the **batch_max** option. When this is done for BSMTP, messages may contain multiple RCPT commands. See chapter 24 for more details.

When one or more addresses are routed to a BSMTP transport by a router that sets up a host list, the name of the first host on the list is available to the transport in the variable **$host**. Here is an example of such a transport and router:

```
begin routers
route_append:
  driver = manualroute
  transport = smtp_appendfile
  route_list = domain.example  batch.host.example

begin transports
smtp_appendfile:
  driver = appendfile
  directory = /var/bsmtp/$host
  batch_max = 1000
  use_bsmtp
  user = exim
```

This causes messages addressed to *domain.example* to be written in BSMTP format to **/var/bsmtp/batch.host.example**, with only a single copy of each message (unless there are more than 1000 recipients).

42.12 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, 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. HELO and EHLO act as RSET; VRFY, EXPN, ETRN and HELP, act as NOOP; QUIT quits.

No policy checking is done for BSMTP input. That is, no ACL is run at anytime. In this respect it is like non-SMTP local input.

If an 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, and the addition or removal of header lines while delivering, happens when a message is received, before it is placed on Exim's queue.

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 receipient addresses in SMTP commands, namely **sender_unqualified_hosts** and **recipient_unqualified_hosts**. In both cases, if an unqualified address is accepted, it is qualified by adding the value of **qualify_domain** or **qualify_recipient**, 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 2822 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 that contains a 'From' line 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 2822 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 Resent- header lines

RFC 2822 makes provision for sets of header lines starting with the string `Resent-` to be added to a message when it is resent by the original recipient to somebody else. These headers are *Resent-Date:*, *Resent-From:*, *Resent-Sender:*, *Resent-To:*, *Resent-Cc:*, *Resent-Bcc:* and *Resent-Message-ID:*. The RFC says:

Resent fields are strictly informational. They MUST NOT be used in the normal processing of replies or other such automatic actions on messages.

This leaves things a bit vague as far as other processing actions such as address rewriting are concerned. Exim treats **Resent-** header lines as follows:

- A *Resent-From:* line that just contains the login id of the submitting user is automatically rewritten in the same way as *From:* (see below).
- If there's a rewriting rule for a particular header line, it is also applied to **Resent-** header lines of the same type. For example, a rule that rewrites *From:* also rewrites *Resent-From:*.
- For local messages, if *Sender:* is removed on input, *Resent-Sender:* is also removed.
- If there are any **Resent-** header lines but no *Resent-Date:*, *Resent-From:*, or *Resent-Message-Id:*, they are added as necessary. It is the contents of *Resent-Message-Id:* (rather than *Message-Id:*) which are included in log lines in this case.
- The logic for adding *Sender:* is duplicated for *Resent-Sender:* when any **Resent-** header lines are present.

43.4 The Bcc: header line

If Exim is called with the **-t** option, to take recipient addresses from a message's header, it removes any *Bcc:* header line that may exist (after extracting its addresses), unless the message has no *To:* or *Cc:*, in which case a *Bcc:* header line with no addresses is left in the message. If **-t** is not present on the command line, any existing *Bcc:* 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, an empty *Bcc:* header is added.

When there is no *To:* or *Cc:*, the presence of an empty *Bcc:* is needed to make the message conform to RFC 822. It is not in fact needed for RFC 2822 conformance, and may be removed in future when RFC 2822 has been around for some time.

43.5 The Date: header line

If a message has no *Date:* header line, Exim adds one, using the current date and time.

43.6 The Delivery-date: header line

Delivery-date: header lines are not part of the standard RFC 2822 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:* header lines from incoming messages.

43.7 The Envelope-to: header line

Envelope-to: header lines are not part of the standard RFC 2822 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:* header lines from incoming messages.

43.8 The From: header line

If an incoming message does not contain a *From:* header line, 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 data 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 line 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.

43.9 The Message-ID: header line

If an incoming message does not contain a *Message-ID:* or *Resent-Message-ID:* header line, Exim adds one to the message. If there are any *Resent-:* headers in the message, it creates *Resent-Message-ID:*. 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 line by setting the **message_id_header_text** and/or **message_id_header_domain** options.

43.10 The Received: header line

A *Received:* header line is added at the start of every message. The contents are defined by the **received_header_text** configuration option, and Exim automatically adds a semicolon and a timestamp to the configured string.

43.11 The Return-path: header line

Return-path: header lines are defined as something an 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:* header lines from incoming messages.

43.12 The Sender: header line

For a locally-originated message from an untrusted user, Exim may remove an existing *Sender:* header line, and it may add a new one. You can modify these actions by setting **local_sender_retain** true or **local_from_check** false. No processing of *Sender:* header lines is done for messages received by TCP/IP or for messages submitted by trusted users.

When a local message is received from an untrusted user and **local_from_check** is true (the default), a check is made to see if the address given in the *From:* header line is the correct (local) sender of the message. The address that is expected has the login name as the local part and the value of **qualify_domain** as the domain. Prefixes and suffixes for the local part can be permitted by setting **local_from_prefix** and **local_from_suffix** appropriately. If *From:* does not contain the correct sender, a *Sender:* line is added to the message.

If you set **local_from_check** false, this checking does not occur. However, the removal of an existing *Sender:* line still happens, unless you also set **local_sender_retain** to be true. It is not possible to set both of these options true at the same time.

43.13 Adding and removing header lines

When a message is delivered, the addition and removal of header lines can be specified on any of the routers and transports, and also in the system filter. Changes specified in the system filter affect all deliveries of a message.

Header changes specified on a router affect all addresses handled by that router, and also any new addresses it generates. If an address passes through several routers, the changes are cumulative. When a message is processed by a transport, the message's original set of header lines 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 the new header lines from **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.example>
```

The user name is obtained from the **-F** command line option if set, or otherwise by looking up the calling user by *getpwuid()* and extracting 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 2822 by quoting all or parts of it if necessary. In addition, if it contains any non-printing characters, it is encoded as described in RFC 2047, which defines a way of including non-ASCII characters in header lines. The setting of **print_topbitchars** controls whether characters with the top bit set (that is, with codes greater than 127) count as printing characters or not.

43.15 Case of local parts

RFC 2822 states that the case of letters in the local parts of addresses cannot be assumed to be non-significant. Exim preserves the case of local parts of addresses, but by default it uses a lower-cased form when it is routing, because on most Unix systems, usernames are in lower case and case-insensitive routing is required. However, any particular router can be made to use the original case for local parts by setting the **caseful_local_part** generic router option.

If you must have mixed-case user names on your system, the best way to proceed, assuming you want case-independent handling of incoming email, is to set up your first router to convert incoming local parts in your domains to the correct case by means of a file lookup. For example:

```
correct_case:
  driver = redirect
  domains = +local_domains
  data = ${lookup{$local_part}cdb\
              {/etc/usercased.cdb}{$value}fail}\
              @$domain
```

For this router, the local part is forced to lower case by the default action (**caseful_local_part** is not set). The lower-cased local part is used to look up a new local part in the correct case. If you then set **caseful_local_part** on any subsequent routers which process your domains, they will operate on local parts with the correct case in a case-sensitive manner.

43.16 Dots in local parts

RFC 2822 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 30. 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.example
```

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. Log files

Exim writes three different logs, referred to as the main log, the reject log, and the panic log:

- The main log 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 recorded in the main log. Some of them are optional, in which case the **log_selector** option controls whether they are included or not. A Perl script called *eximstats* that does simple analysis of main log files is provided in the Exim distribution (see section 45.6).
- The reject log records information from messages that are rejected as a result of a configuration option (that is, for policy reasons). If the message's header has been read at the time the log is written, its contents are written to this log, following a copy of the one-line message that is written to the main log. You can use the reject log to check that your policy controls are working correctly.
- When certain serious errors occur, Exim writes entries to its panic log. If the error is sufficiently disastrous, Exim bombs out afterwards. Panic log entries are usually written 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. When Exim cannot open its panic log, it tries as a last resort to write to the system log (syslog). This is opened with LOG_PID+LOG_CONS and the facility code of LOG_MAIL. The message itself is written at priority LOG_CRIT.

Every log line starts with a timestamp, in the format shown in this example:

```
2001-09-16 16:09:47 SMTP connection from [127.0.0.1] closed by QUIT
```

By default, the timestamps are in the local timezone. There are two ways of changing this:

- You can set the **timezone** option to a different time zone; in particular, if you set

  ```
  timezone = UTC
  ```

  the timestamps will be in UTC (aka GMT).
- If you set **log_timezone** true, the time zone is added to the timestamp, for example:

  ```
  2003-04-25 11:17:07 +0100 Start queue run: pid=12762
  ```

44.1 Where the logs are written

The logs may be written to local files, or to syslog, or both. However, it should be noted that many syslog implementations use UDP as a transport, and are therefore unreliable in the sense that messages are not guaranteed to arrive at the loghost, nor is the ordering of messages necessarily maintained. It has also been reported that on large log files (tens of megabytes) you may need to tweak syslog to prevent it syncing the file with each write – on Linux this has been seen to make syslog take 90% plus of CPU time.

The destination for Exim's logs is configured by setting LOG_FILE_PATH in **Local/Makefile** or by setting **log_file_path** in the run time configuration. This latter string is expanded, so it can contain, for example, references to the host name:

```
log_file_path = /var/log/$primary_hostname/exim_%slog
```

It is generally advisable, however, to set the string in **Local/Makefile** rather than at run time, because then the setting is available right from the start of Exim's execution. Otherwise, if there's something it wants to log before it has read the configuration file (for example, an error in the configuration file) it will not use the path you want, and may not be able to log at all.

The value of LOG_FILE_PATH or **log_file_path** is a colon-separated list, currently limited to at most two items. This is one option where the facility for changing a list separator may not be used. The list must always be colon-separated. If an item in the list is 'syslog' then syslog is used; otherwise the item must either be an absolute path, containing `%s` at the point where 'main', 'reject', or 'panic' is to be inserted, or be empty, implying the use of a default path.

When Exim encounters an empty item in the list, it searches the list defined by LOG_FILE_PATH, and uses the first item it finds that is neither empty nor 'syslog'. This means that an empty item in **log_file_path** can be used to mean 'use the path specified at build time'. It no such item exists, log files are written in the **log** subdirectory of the spool directory. This is equivalent to the setting:

```
log_file_path = $spool_directory/log/%slog
```

If you do not specify anything at build time or run time, that is where the logs are written.

A log file path may also contain `%D` if datestamped log file names are in use – see section 44.3 below.

Here are some examples of possible settings:

`LOG_FILE_PATH=syslog`	syslog only
`LOG_FILE_PATH=:syslog`	syslog and default path
`LOG_FILE_PATH=syslog : /usr/log/exim_%s`	syslog and specified path
`LOG_FILE_PATH=/usr/log/exim_%s`	specified path only

If there are more than two paths in the list, the first is used and a panic error is logged.

44.2 Logging to local files that are periodically 'cycled'

Some operating systems provide centralized and standardised methods for cycling log files. For those that do not, a utility script called *exicyclog* is provided (see section 45.5). This renames and compresses the main and reject logs each time it is called. The maximum number of old logs to keep can be set. It is suggested this script is run as a daily *cron* job.

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.

44.3 Datestamped log files

Instead of cycling the main and reject log files by renaming them periodically, some sites like to use files whose names contain a datestamp, for example, **mainlog-20031225**. The datestamp is in the form **yyyymmdd**. Exim has support for this way of working. It is enabled by setting the **log_file_path** option to a path that includes `%D` at the point where the datestamp is required. For example:

```
log_file_path = /var/spool/exim/log/%slog-%D
log_file_path = /var/log/exim-%s-%D.log
log_file_path = /var/spool/exim/log/%D-%slog
```

As before, `%s` is replaced by 'main' or 'reject'; the following are examples of names generated by the above examples:

```
/var/spool/exim/log/mainlog-20021225
/var/log/exim-reject-20021225.log
/var/spool/exim/log/20021225-mainlog
```

When this form of log file is specified, Exim automatically switches to new files at midnight. It does not make any attempt to compress old logs; you will need to write your own script if you require this. You should not run *exicyclog* with this form of logging.

The location of the panic log is also determined by **log_file_path**, but it is not datestamped, because rotation of the panic log does not make sense. When generating the name of the panic log, `%D` is removed from the string. In addition, if it immediately follows a slash, a following non-alphanumeric character is removed; otherwise a preceding non-alphanumeric character is removed. Thus, the three examples above would give these panic log names:

```
/var/spool/exim/log/paniclog
/var/log/exim-panic.log
/var/spool/exim/log/paniclog
```

44.4 Logging to syslog

The use of syslog does not change what Exim logs or the format of its messages, except in one respect. If **syslog_timestamp** is set false, the timestamps on Exim's log lines are omitted when these lines are sent to syslog. Apart from that, the same strings are written to syslog as to log files. The syslog 'facility' is set to LOG_MAIL, and the program name to 'exim' by default, but you can change these by setting the **syslog_facility** and **syslog_processname** options, respectively. If Exim was compiled with SYSLOG_LOG_PID set in **Local/Makefile** (this is the default in **src/EDITME**), then, on systems that permit it (all except ULTRIX), the LOG_PID flag is set so that the *syslog()* call adds the pid as well as the time and host name to each line. The three log streams are mapped onto syslog priorities as follows:

mainlog is mapped to LOG_INFO

rejectlog is mapped to LOG_NOTICE

paniclog is mapped to LOG_ALERT

Many log lines are written to both *mainlog* and *rejectlog*, so there will be duplicates if these are routed by syslog to the same place.

Exim's log lines can sometimes be very long, and some of its *rejectlog* entries contain multiple lines when headers are included. To cope with both these cases, entries written to syslog are split into separate *syslog()* calls at each internal newline, and also after a maximum of 870 data characters. (This allows for a total syslog line length of 1024, when additions such as timestamps are added.) If you are running a syslog replacement that can handle lines longer than the 1024 characters allowed by RFC 3164, you should set

```
SYSLOG_LONG_LINES=yes
```

in **Local/Makefile** before building Exim. That stops Exim from splitting long lines, but it still splits at internal newlines in *reject* log entries.

To make it easy to re-assemble split lines later, each component of a split entry starts with a string of the form '[<*n*>/<*m*>]' or '[<*n*>\<*m*>]' where <*n*> is the component number and <*m*> is the total number of components in the entry. The / delimiter is used when the line was split because it was too long; if it was split because of an internal newline, the \ delimiter is used. For example, supposing the length limit to be 70 instead of 1000, the following would be the result of a typical rejection message to *mainlog* (LOG_INFO), each line in addition being preceded by the time, host name, and pid as added by syslog:

```
[1/3] 2002-09-16 16:09:43 16RdAL-0006pc-00 rejected from [127.0.0.1] (ph10):
[2/3]  syntax error in 'From' header when scanning for sender: missing or ma
[3/3] lformed local part in "<>" (envelope sender is <ph10@cam.example>)
```

The same error might cause the following lines to be written to 'rejectlog' (LOG_NOTICE):

```
[1/14] 2002-09-16 16:09:43 16RdAL-0006pc-00 rejected from [127.0.0.1] (ph10):
[2/14]  syntax error in 'From' header when scanning for sender: missing or ma
[3\14] lformed local part in "<>" (envelope sender is <ph10@cam.example>)
[4\14] Recipients: ph10@some.domain.cam.example
[5\14] P Received: from [127.0.0.1] (ident=ph10)
[6\14]        by xxxxx.cam.example with smtp (Exim 4.00)
[7\14]        id 16RdAL-0006pc-00
[8\14]        for ph10@cam.example; Mon, 16 Sep 2002 16:09:43 +0100
[9\14] F From: <>
[10\14]   Subject: this is a test header
[11\14]   X-something: this is another header
[12\14] I Message-Id: <E16RdAL-0006pc-00@xxxxx.cam.example>
[13\14] B Bcc:
[14/14]   Date: Mon, 16 Sep 2002 16:09:43 +0100
```

Log lines that are neither too long nor contain newlines are written to syslog without modification.

If only syslog is being used, the Exim monitor is unable to provide a log tail display, unless syslog is routing *mainlog* to a file on the local host and the environment variable EXIMON_LOG_FILE_PATH is set to tell the monitor where it is.

44.5 Log line flags

One line is written to the main log for each message received, and for each successful, unsuccessful, and delayed delivery. These lines can readily be picked out by the distinctive two-character flags that immediately follow the timestamp. The flags are:

`<=`	message arrival
`=>`	normal message delivery
`->`	additional address in same delivery
`*>`	delivery suppressed by **-N**
`**`	delivery failed; address bounced
`==`	delivery deferred; temporary problem

44.6 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 basic example below, which is split over several lines in order to fit it on the page:

```
2002-10-31 08:57:53 16ZCW1-0005MB-00 <= kryten@dwarf.fict.example
  H=mailer.fict.example [192.168.123.123] U=exim
  P=smtp S=5678 id=<incoming message id>
```

The address immediately following '<=' is the envelope sender address. A bounce message is shown with the sender address '<>', and if it is locally generated, this is followed by an item of the form

```
R=<message id>
```

which is a reference to the message that caused the bounce to be sent.

For messages from other hosts, 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, 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 text like these examples:

```
H=(10.21.32.43) [192.168.8.34]
H=([10.21.32.43]) [192.168.8.34]
```

This can be confusing. Only the final address in square brackets can be relied on.

For locally generated messages (that is, messages not received over TCP/IP), 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. This is set to ‘asmtp’ for messages received from hosts which have authenticated themselves using the SMTP AUTH command. In this case there is an additional item A= followed by the name of the authenticator that was used. If an authenticated identification was set up by the authenticator’s **server_set_id** option, this is logged too, separated by a colon from the authenticator name.

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).

The **log_selector** option can be used to request the logging of additional data when a message is received. See section 44.15 below.

44.7 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:

```
2002-10-31 08:59:13 16ZCW1-0005MB-00 => marv <marv@hitch.fict.example>
  R=localuser T=local_delivery
2002-10-31 09:00:10 16ZCW1-0005MB-00 => monk@holistic.fict.example
  R=dnslookup T=remote_smtp H=holistic.fict.example [192.168.234.234]
```

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. The R and T fields record the router and transport that were used to process the address.

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 more than one address is included in a single delivery (for example, two SMTP RCPT commands in one transaction) 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.

The generation of a reply message by a filter file gets logged as a ‘delivery’ to the addressee, preceded by ‘>’.

The **log_selector** option can be used to request the logging of additional data when a message is delivered. See section 44.15 below.

44.8 Discarded deliveries

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

```
2002-12-10 00:50:49 16auJc-0001UB-00 => discarded
  <low.club@bridge.example> R=userforward
```

is written, to record why no deliveries are logged. When a message is discarded because it is aliased to ':blackhole:' the log line is like this:

```
1999-03-02 09:44:33 10HmaX-0005vi-00 => :blackhole:
  <hole@nowhere.example> R=blackhole_router
```

44.9 Deferred deliveries

When a delivery is deferred, a line of the following form is logged:

```
2002-12-19 16:20:23 16aiQz-0002Q5-00 == marvin@endrest.example
  R=dnslookup 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

```
2002-12-19 16:20:23 16aiQz-0002Q5-00 Failed to connect to
  mail1.endrest.example [192.168.239.239]: 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 setting an appropriate value in **log_selector**.

44.10 Delivery failures

If a delivery fails because an address cannot be routed, a line of the following form is logged:

```
1995-12-19 16:20:23 0tRiQz-0002Q5-00 ** jim@trek99.example
  <jim@trek99.example>: unknown mail domain
```

If a delivery fails at transport time, the router and transport are shown, and the response from the remote host is included, as in this example:

```
2002-07-11 07:14:17 17SXDU-000189-00 ** ace400@pb.example R=dnslookup
  T=remote_smtp: SMTP error from remote mailer after
  RCPT TO:<ace400@pb.example>: host pbmail3.py.example
  [192.168.63.111]: 553 5.3.0 <ace400@pb.example>...
  Addressee unknown
```

The log lines for all forms of delivery failure are flagged with `**`.

44.11 Fake deliveries

If a delivery does not actually take place because the **-N** option has been used to suppress it, a normal delivery line is written to the log, except that '=>' is replaced by '*>'.

44.12 Completion

A line of the form

```
2002-10-31 09:00:11 16ZCW1-0005MB-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.

44.13 Summary of Fields in Log Lines

A summary of the field identifiers that are used in log lines is shown in the following table:

A	authenticator name (and optional id)
C	SMTP confirmation on delivery
DN	DN from peer certificate
F	sender address (on delivery lines)
H	host name and IP address
id	message id for incoming message
P	protocol for incoming message
R	on <= lines: reference for local bounce on => lines: router name
S	size of message
ST	shadow transport name
T	on <= lines: message subject (topic) on => lines: transport name
U	local user or RFC 1413 identity
X	TLS cipher suite

44.14 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 routing or local delivery, and the time to retry has not yet arrived. This message is not written to an individual message log file unless it happens during the first delivery attempt.
- *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.

44.15 Reducing or increasing what is logged

By setting the **log_selector** global option, you can disable some of Exim's default logging, or you can request additional logging. The value of **log_selector** is made up of names preceded by plus or minus characters. For example:

```
log_selector = +arguments -retry_defer
```

The list of optional log items is in the following table, with the default selection marked by asterisks:

address_rewrite	address rewriting
all_parents	all parents in => lines
arguments	command line arguments
*connection_reject	connection rejections
*delay_delivery	immediate delivery delayed (message queued)
delivery_size	add S=nnn to => lines
*dnslist_defer	defers of DNS list (aka RBL) lookups
*etrn	ETRN commands
*host_lookup_failed	as it says
incoming_interface	incoming interface on <= lines
incoming_port	incoming port on <= lines
*lost_incoming_connection	as it says (includes timeouts)
*queue_run	start and end queue runs
received_recipients	recipients on <= lines
received_sender	sender on <= lines
*rejected_header	header contents on reject log
*retry_defer	'retry time not reached'

`sender_on_delivery`	add sender to => lines
`*size_reject`	rejection because too big
`*skip_delivery`	‘message is frozen’, ‘spool file is locked’
`smtp_confirmation`	SMTP confirmation on <= lines
`smtp_connection`	SMTP connections
`smtp_protocol_error`	SMTP protocol errors
`smtp_syntax_error`	SMTP syntax errors
`subject`	contents of *Subject:* on <= lines
`*tls_cipher`	TLS cipher suite on <= and => lines
`tls_peerdn`	TLS peer DN on <= and => lines
`all`	all of the above

More details on each of these items follows:

- **address_rewrite**: This applies both to global rewrites and per-transport rewrites.
- **all_parents**: Normally only the original and final addresses are logged on delivery lines; with this selector, intermediate parents are given in parentheses between them.
- **arguments**: This causes Exim to write the arguments with which it was called to the main log, preceded by the current working directory. This is a debugging feature, added to make it easier to find out how certain MUAs call **/usr/sbin/sendmail**. The logging does not happen if Exim has given up root privilege because it was called with the **-C** or **-D** options. Arguments that are empty or that contain whitespace are quoted. Non-printing characters are shown as escape sequences. This facility cannot log unrecognized arguments, because the arguments are checked before the configuration file is read. The only way to log such cases is to interpose a script such as **util/logargs.sh** between the caller and Exim.
- **connection_reject**: A log entry is written whenever an incoming SMTP connection is rejected, for whatever reason.
- **delay_delivery**: A log entry is written whenever a delivery process is not started for an incoming message because the load is too high or too many messages were received on one connection. Logging does not occur if no delivery process is started because **queue_only** is set or **-odq** was used.
- **delivery_size**: For each delivery, the size of message delivered is added to the ‘=>’ line, tagged with S=.
- **dnslist_defer**: A log entry is written if an attempt to look up a host in a DNS black list suffers a temporary error.
- **etrn**: Every legal ETRN command that is received is logged, before the ACL is run to determine whether or not it is actually accepted. An invalid ETRN command, or one received within a message transaction is not logged by this selector (see **smtp_syntax_error** and **smtp_protocol_error**).
- **host_lookup_failed**: When a lookup of a host’s IP addresses fails to find any addresses, or when a lookup of an IP address fails to find a host name, a log line is written. This logging does not apply to direct DNS lookups when routing email addresses, but it does apply to ‘byname’ lookups.
- **incoming_interface**: The interface on which a message was received is added to the ‘<=’ line as an IP address in square brackets, tagged by I= and followed by a colon and the port number.
- **incoming_port**: The remote port number from which a message was received is added to log entries and *Received:* header lines, following the IP address in square brackets, and separated from it by a colon. This is implemented by changing the value that is put in the **$sender_fullhost** and **$sender_rcvhost** variables. Recording the remote port number has become more important with the widening use of NAT (see RFC 2505).

- **lost_incoming_connection**: A log line is written when an incoming SMTP connection is unexpectedly dropped.
- **queue_run**: The start and end of every queue run are logged.
- **received_recipients**: 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. Recipients that were discarded by an ACL for MAIL or RCPT do not appear in the list.
- **received_sender**: 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 **received_recipients** is also set).
- **rejected_header**: If a message's header has been received at the time a rejection is written to the reject log, the complete header is added to the log. Header logging can be turned off individually for messages that are rejected by the *local_scan()* function (see section 38.2).
- **retry_defer**: A log line is written if a delivery is deferred because a retry time has not yet been reached. However, this 'retry time not reached' message is always omitted from individual message logs after the first delivery attempt.
- **sender_on_delivery**: The message's sender address is added to every delivery and bounce line, tagged by F= (for 'from').
- **size_reject**: A log line is written whenever a message is rejected because it is too big.
- **skip_delivery**: A log line is written whenever a message is skipped during a queue run because it is frozen or because another process is already delivering it.
- **smtp_confirmation**: The response to the final '.' in the SMTP dialogue for outgoing messages is added to delivery log lines in the form 'C="<*text*>"'. A number of MTAs (including Exim) return an identifying string in this response.
- **smtp_connection**: A log line is written whenever an SMTP connection is established or closed. (By contrast, **lost_incoming_connection** applies only when the closure is unexpected.) This applies to connections from local processes that use **-bs** as well as to TCP/IP connections. If a connection is dropped in the middle of a message, a log line is always written, whether this selector is set or not, but otherwise nothing is written at the start and end of connections unless this selector is enabled. For TCP/IP connections to an Exim daemon, the current number of connections is included in the log message for each new connection, but note that the count is reset if the daemon is restarted.
- **smtp_protocol_error**: A log line is written for every SMTP protocol error encountered.
- **smtp_syntax_error**: A log line is written for every SMTP syntax error encountered. 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.
- **subject**: The subject of the message is added to the arrival log line, preceded by 'T=' (T for 'topic', since S is already used for 'size').
- **tls_cipher**: When a message is sent or received over an encrypted connection, the cipher suite used is added to the log line, preceded by X=.
- **tls_peerdn**: When a message is sent or received over an encrypted connection, and a certificate is supplied by the remote host, the peer DN is added to the log line, preceded by DN=.

44.16 Message log

In addition to the general 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. Each message log contains copies of the log lines that apply to the message. This makes it easier to inspect the status of an individual message without having to search the main log. A message 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 disk very quickly.

On a heavily loaded system, it may be desirable to disable the use of per-message logs, in order to reduce disk I/O. This can be done by setting the **message_logs** option false.

45. Exim utilities

A number of utility scripts and programs are supplied with Exim and are described in this chapter. There is also the Exim Monitor, which is covered in the next chapter. The utilities described here are:

45.1	*exiwhat*	list what Exim processes are doing
45.2	*exiqgrep*	grep the queue
45.3	*exiqsumm*	summarize the queue
45.4	*exigrep*	search the main log
45.5	*exicyclog*	cycle (rotate) log files
45.6	*eximstats*	extract statistics from the log
45.7	*exim_checkaccess*	check address acceptance from given IP
45.8	*exim_dbmbuild*	build a DBM file
45.9	*exinext*	extract retry information
45.10	*exim_dumpdb*	dump a hints database
45.10	*exim_tidydb*	clean up a hints database
45.10	*exim_fixdb*	patch a hints database
45.11	*exim_lock*	lock a mailbox file

45.1 Finding out what Exim processes are doing (exiwhat)

On operating systems that can restart a system call after receiving a signal (most modern OS), an Exim process responds to the SIGUSR1 signal by writing a line describing what it is doing to the file **exim-process.info** in the Exim spool directory. The *exiwhat* script sends the signal to all Exim processes it can find, having first emptied the file. 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.

Unfortunately, the *ps* command which *exiwhat* uses to find Exim processes varies in different operating systems. 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

An example of typical output from *exiwhat* is

```
  164 daemon: -q1h, listening on port 25
10483 running queue: waiting for 0tAycK-0002ij-00 (10492)
10492 delivering 0tAycK-0002ij-00 to mail.ref.example [10.19.42.42]
  (editor@ref.example)
10592 handling incoming call from [192.168.243.242]
10628 accepting a local non-SMTP message
```

The first number in the output line is the process number. The third line has been split here, in order to fit it on the page.

45.2 Selective queue listing (exiqgrep)

This utility is a Perl script contributed by Matt Hubbard. It runs

```
exim -bpu
```

to obtain a queue listing with undelivered recipients only, and then greps the output to select messages that match given criteria. The following selection options are available:

-f<*regex*>
: Match the sender address. The field that is tested is enclosed in angle brackets, so you can test for bounce messages with

```
exiqgrep -f '^<>$'
```

-r<*regex*>
: Match a recipient address. The field that is tested is not enclosed in angle brackets.

-s<*regex*>
: Match against the size field.

-y<*seconds*>
: Match messages that are younger than the given time.

-o<*seconds*>
: Match messages that are older than the given time.

-z
: Match only frozen messages.

-x
: Match only non-frozen messages.

The following options control the format of the output:

-c
: Display only the count of matching messages.

-l
: Long format – display the full message information as output by Exim. This is the default.

-i
: Display message ids only.

-b
: Brief format – one line per message.

-R
: Display messages in reverse order.

There is one more option, **-h**, which outputs a list of options.

45.3 Summarising the queue (exiqsumm)

The *exiqsumm* utility is a Perl script which reads the output of *exim -bp* and produces a summary of the messages on the queue. Thus, you use it by running a command such as

```
exim -bp | exiqsumm
```

The output consists of one line for each domain that has messages waiting for it, as in the following example:

```
   3   2322   74m   66m  msn.com.example
```

Each line lists the number of messages for a domain, their total volume, and the length of time that the oldest and the newest messages have been waiting. A summary line is output at the end. By default the output is sorted on the domain name, but *exiqsumm* has the options **-a** and **-c**, which cause the output to be sorted by oldest message and by count of messages, respectively.

The output of *exim -bp* contains the original addresses in the message, so this also applies to the output from *exiqsumm*. No domains from addresses generated by aliasing or forwarding are included (unless the **one_time** option of the **redirect** router has been used to convert them into 'top level' addresses).

45.4 Extracting specific information from the log (exigrep)

The *exigrep* utility is a Perl script that searches one or more main log files for entries that match a given pattern. When it finds a match, it extracts all the log entries for the relevant message, not just those that match the pattern. Thus, *exigrep* can extract complete log entries for a given message, or all mail for a given user, or for 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. The pattern match is case-insensitive. If no file names are given on the command line, the standard input is read.

If the location of a *zcat* command is known from the definition of ZCAT_COMMAND in **Local/Makefile**, *exigrep* automatically passes any file whose name ends in COMPRESS_SUFFIX through *zcat* as it searches it.

45.5 Cycling log files (exicyclog)

The *exicyclog* script can be used to cycle (rotate) *mainlog* and *rejectlog* files. This is not necessary if only syslog is being used, or if you are using log files with datestamps in their names (see section 44.3). Some operating systems have their own standard mechanisms for log cycling, and these can be used instead of *exicyclog* if preferred.

Each time *exicyclog* is run the file names 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. Reject logs are handled similarly.

If no **mainlog** file exists, the script does nothing. Files that ‘drop off’ the end are deleted. All files with numbers greater than 01 are compressed, using a compression command which is configured by the COMPRESS_COMMAND setting in **Local/Makefile**. It is usual to run *exicyclog* daily from a root **crontab** entry of the form

```
1 0 * * * su exim -c /usr/exim/bin/exicyclog
```

assuming you have used the name ‘exim’ for the Exim user. You can run *exicyclog* as root if you wish, but there is no need.

45.6 Mail statistics (eximstats)

A Perl script called *eximstats* is provided for extracting statistical information from log files. The output is either plain text, or HTML. Exim log files are also suported by the *Lire* system produced by the LogReport Foundation (**http://www.logreport.org**).

The *eximstats* script has been hacked about quite a bit over time. The latest version is the result of some extensive revision by Steve Campbell. A lot of information is given 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. For example:

```
eximstats -nr /var/spool/exim/log/mainlog.01
```

By default, *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. Similar information, based on email addresses or domains instead of hosts can be requested by means of various options. For messages delivered and received locally, similar statistics are also 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 queue 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. A bounce message 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, without being processed (for example, for aliasing or forwarding) locally.

The options for *eximstats* are as follows:

-bydomain
: The 'league tables' are computed on the basis of the superior domains of the sending hosts instead of the sending and receiving hosts. This option may be combined with **-byhost** and/or **-byemail**.

-byedomain
: This is a synonym for **-byemaildomain**.

-byemail
: The 'league tables' are computed on the basis of complete email addresses, instead of sending and receiving hosts. This option may be combined with **-byhost** and/or **-bydomain**.

-byemaildomain
: The 'league tables' are computed on the basis of the sender's email domain instead of the sending and receiving hosts. This option may be combined with **-byhost**, **-bydomain**, or **-byemail**.

-byhost
: The 'league tables' are computed on the basis of sending and receiving hosts. This is the default option. It may be combined with **-bydomain** and/or **-byemail**.

-cache Cache results of *timegm()* lookups. This results in a significant speedup when processing hundreds of thousands of messages, at a cost of increasing the memory utilisation.

-chartdir<*dir*>
: When **-charts** is specified, create the charts in the directory <*dir*>.

-chartrel<*dir*>
: When **-charts** is specified, this option specifies the relative directory for the `img src=` tags from where to include the charts.

-charts Create graphical charts to be displayed in HTML output. This requires the `GD`, `GDTextUtil`, and `GDGraph` Perl modules, which can be obtained from **http://www.cpan.org/modules/01modules.index.html**.

To install these, download and unpack them, then use the normal Perl installation procedure:

```
perl Makefile.PL
make
make test
make install
```

-d This is a debug flag. It causes *eximstats* to output the *eval()*'d parser to the standard output, which makes it easier to trap errors in the eval section. Remember to add one to the line numbers to allow for the title.

-help Show help information about *eximstats*' options.

-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. Valid values are 0, 1, 2, 3, 5, 10, 15, 20, 30 or 60, so **-h2** sets an interval of 30 minutes, and the default is equivalent to **-h1**.

-html Output the results in HTML instead of plain text.

-merge This option causes *eximstats* to merge old reports into a combined report. When this option is used, the input files must be outputs from previous calls to *eximstats*, not raw log files. For example, you could produce a set of daily reports and a weekly report by commands such as

```
eximstats mainlog.sun > report.sun.txt
eximstats mainlog.mon > report.mon.txt
eximstats mainlog.tue > report.tue.txt
eximstats mainlog.wed > report.wed.txt
eximstats mainlog.thu > report.thu.txt
eximstats mainlog.fri > report.fri.txt
eximstats mainlog.sat > report.sat.txt
eximstats -merge -html report.*.txt > weekly_report.html
```

You can merge text or html reports and output the results as text or html. You can use all the normal *eximstats* output options, but only data included in the original reports can be shown. When merging reports, some loss of accuracy may occur in the 'league tables', towards the ends of the lists. The order of items in the 'league tables' may vary when the data volumes round to the same value.

-ne Suppress the display of information about failed deliveries (errors).

-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.example =>
  H=out.host A=to@else.where.example
```

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.

-nt Suppress the statistics about delivery by transport.

-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.

-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.

-t_remote_users
Include remote users in the ‘top’ listings.

45.7 Checking access policy (exim_checkaccess)

The **-bh** command line argument allows you to run a fake SMTP session with debugging output, in order to check what Exim is doing when it is applying policy controls to incoming SMTP mail. However, not everybody is sufficiently familiar with the SMTP protocol to be able to make full use of **-bh**, and sometimes you just want to answer the question *Does this address have access?* without bothering with any further details.

The *exim_checkaccess* utility is a ‘packaged’ version of **-bh**. It takes two arguments, an IP address and an email address:

```
exim_checkaccess 10.9.8.7 A.User@a.domain.example
```

The utility runs a call to Exim with the **-bh** option, to test whether the given email address would be accepted in a RCPT command in a TCP/IP connection from the host with the given IP address. The output of the utility is either the word ‘accepted’, or the SMTP error response, for example:

```
Rejected:
  550 Relay not permitted
```

When running this test, the utility uses <> as the envelope sender address for the MAIL command, but you can change this by providing additional options. These are passed directly to the Exim command. For example, to specify that the test is to be run with the sender address *himself@there.example* you can use:

```
exim_checkaccess 10.9.8.7 A.User@a.domain.example \
                 -f himself@there.example
```

Note that these additional Exim command line items must be given after the two mandatory arguments.

45.8 Making DBM files (exim_dbmbuild)

The *exim_dbmbuild* program reads an input file containing keys and data in the format used by the **lsearch** lookup (see section 9.2). It writes a DBM file using the lower-cased alias names as keys and the remainder of the information as data. The lower-casing can be prevented by calling the program with the **-nolc** option.

A terminating zero is included as part of the key string. This is expected by the **dbm** lookup type. However, if the option **-nozero** is given, *exim_dbmbuild* creates files without terminating zeroes in either the key strings or the data strings. The **dbmnz** lookup type can be used with such files.

The program requires two arguments: the name of the input file (which can be a single hyphen to indicate the standard input), and the name of the output file. It creates the output under a temporary name, and then renames it if all went well. If the native DB interface is in use (USE_DB is set in a compile-time configuration file – this is common in free versions of Unix) the two file names must be different, because in this mode the Berkeley DB functions create a single output file using exactly the name given. For example,

```
exim_dbmbuild /etc/aliases /etc/aliases.db
```

reads the system alias file and creates a DBM version of it in **/etc/aliases.db**.

In systems that use the *ndbm* routines (mostly proprietary versions of Unix), two files are used, with the suffixes **.dir** and **.pag**. In this environment, the suffixes are added to the second argument of *exim_dbmbuild*, so it can be the same as the first. This is also the case when the Berkeley functions are used in compatibility mode (though this is not recommended), because in that case it adds a **.db** suffix to the file name.

If a duplicate key is encountered, the program outputs a warning, and when it finishes, its return code is 1 rather than zero, unless the **-noduperr** option is used. By default, only the first of a set of duplicates is used – this makes it compatible with **lsearch** lookups. There is an option **-lastdup** which causes it to use the data for the last duplicate instead. There is also an option **-nowarn**, which stops it listing duplicate keys to **stderr**. For other errors, where it doesn't actually make a new file, the return code is 2.

45.9 Finding individual retry times (exinext)

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 for the hosts or for the domain. At present, the retry information is obtained by running *exim_dumpdb* (see below) and post-processing the output. For example:

```
exinext piglet@milne.fict.example
kanga.milne.fict.example:192.168.8.1 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.example:192.168.8.3 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 part, without a domain, and it will give any retry information for that local part in your default domain. A message id can be used 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 efficient, but then it isn't expected to be run very often.

45.10 Hints database maintenance (exim_dumpdb, exim_fixdb, exim_tidydb)

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
- *wait-<transport name>*: databases of information about messages waiting for remote hosts
- *misc*: other hints data (for example, for serializing ETRN runs)

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.example:192.168.242.242 146 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 R, or T, depending on whether it refers to a 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 (unless **no_retry_include_ip_address** is set on the **smtp** transport). 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 *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.

The *exim_tidydb* utility program is used to tidy up the contents of the hints 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
```

Both the *wait-xxx* and *retry* databases contain items that involve message ids. In the former these appear as data in records keyed by host – they were messages that were waiting for that host – and in the latter they are the keys for retry information for messages that have suffered certain types of error. When *exim_tidydb* is run, a check is made to ensure that message ids in database records are those of messages that are still on the queue. Message ids for messages that no longer exist are removed from *wait-xxx* records, and if this leaves any records empty, they are deleted. For the *retry* database, records whose keys are non-existent message ids are removed. The *exim_tidydb* utility outputs comments on the standard output whenever it removes information from the database.

Removing records from a DBM file does not normally make the file smaller, but all the common DBM libraries are able to re-use the space that is released. It is therefore suggested that *exim_tidydb* be run periodically on all the hints databases, but at a quiet time of day, because 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.

45.11 Mailbox maintenance (exim_lock)

The *exim_lock* utility locks a mailbox file using the same algorithm as Exim. For a discussion of locking issues, see section 25.3. *Exim_lock* 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.

-flock Use *flock()* locking on the open mailbox, provided the operating system supports it.

-interval

This must be followed by a number, which is a number of seconds; it sets the interval to sleep between retries (default 3).

-lockfile
: Create a lock file before opening the mailbox.

-mbx Lock the mailbox using MBX rules.

-q Suppress verification output.

-retries
: This must be followed by a number; it sets the number of times to try to get the lock (default 10).

-restore_time
: This option causes **exim_lock** to restore the modified and read times to the the locked file before exiting. This allows you to access a locked mailbox (for example, to take a backup copy) without disturbing the times that the user subsequently sees.

-timeout
: This must be followed by a number, which is a number of seconds; it sets a timeout to be used with a blocking *fcntl()* lock. If it is not set (the default), a non-blocking call is used.

-v Generate verbose output.

If none of **-fcntl**, **-flock**, **-lockfile** or **-mbx** are given, the default is to create a lock file and also to use *fcntl()* locking on the mailbox, which is the same as Exim's default. The use of **-flock** or **-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 can be used with either or both of **-fcntl** or **-flock**. It assumes **-fcntl** by default. MBX locking 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.

46. 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 the command line to perform any actions requested.

46.1 Running the monitor

The monitor is started by running the script called *eximon*. This is a shell script that sets up a number of environment variables, and then runs the binary called **eximon.bin**. The default 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. If EXIMON_LOG_FILE_PATH is set in the environment, it overrides the Exim log file configuration. This makes it possible to have *eximon* tailing log data that is written to syslog, provided that MAIL.INFO syslog messages are routed to a file on the local host.

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 queue, and to operate on them, *eximon* must either be run as root or by an admin user.

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, with two more action buttons. The following sections describe these different parts of the display.

46.2 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 disk 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%'. This chart is configured by setting SIZE_STRIPCHART and (optionally) SIZE_STRIPCHART_NAME in the **Local/eximon.conf** file.

46.3 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**.

46.4 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. To save space on the screen, the timestamp on each log line is shortened by removing the date and, if **log_timezone** is set, the timezone. The log tail is not available when the only destination for logging data is syslog, unless the syslog lines are routed to a local file whose name is passed to *eximon* via the EXIMON_LOG_FILE_PATH environment variable.

The log sub-window has a scroll bar at its lefthand side which can be used to move back to look at earlier text, and the up and down arrow keys also have a scrolling effect. The amount of log that is kept depends on the setting of LOG_BUFFER in **Local/eximon.conf**, which specifies the amount of memory to use. When this is full, the earlier 50% of data is discarded – this is much more efficient than throwing it away line by line. The sub-window also has a horizontal scroll bar for accessing the ends of long log lines. This is the only means of horizontal scrolling; the right and left arrow keys are not available. 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 a forward 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 typed 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.

46.5 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.example* 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 do not 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 bounce 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.

46.6 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 48 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 run time.
- *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 bounce message 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 a bounce message.
- *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 you want to set an empty sender (as in bounce messages), you must specify it as '<>'. Otherwise, if the address is not qualified and the QUALIFY_DOMAIN parameter is set in **Local/eximon.conf**, the address is qualified with that domain.

When a delivery is forced, a window showing the **-v** output is displayed. In other cases when a call to Exim is made, if there is 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. However, if you set ACTION_OUTPUT=yes in

Local/eximon.conf, a window showing the Exim command is always opened, even if no output is generated.

The queue display 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 of the display after one of these actions.

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.

47. 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.

47.1 Root privilege

The Exim binary is normally setuid to root, which means that it gains root privilege (runs as root) when it starts execution. In some special cases (for example, when the daemon is not in use and there are no local deliveries), it may be possible to run Exim setuid to some user other than root. This is discussed in the next section. However, in most installations, root privilege is required for two things:

- To set up a socket connected to the standard SMTP port (25) when initialising the listening daemon. If Exim is run from *inetd*, this privileged action is not required.
- To be able to change uid and gid in order to read users’ **.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. For this reason, a user and group for Exim to use must be defined in **Local/Makefile**. These are known as ‘the Exim user’ and ‘the Exim group’. Their values can be changed by the run time configuration, though this is not recommended. Often a user called *exim* is used, but some sites use *mail* or another user name altogether.

Exim uses *setuid()* whenever it gives up root privilege. This is a permanent abdication; the process cannot regain root afterwards. Prior to release 4.00, *seteuid()* was used in some circumstances, but this is no longer the case.

After a new Exim process has interpreted its command line options, it changes uid and gid in the following cases:

- If the **-C** option is used to specify an alternate configuration file, or if the **-D** option is used to define macro values for the configuration, and the calling process is not running as root or the Exim user, the uid and gid are changed to those of the calling process.
- If the expansion test option (**-be**) or one of the filter testing options (**-bf** or **-bF**) are used, the uid and gid are changed to those of the calling process.
- If the process is not a daemon process or a queue runner process or a delivery process or a process for testing address routing (started with **-bt**), the uid and gid are changed to the Exim user and group. This means that Exim always runs under its own uid and gid when receiving messages. This also applies when testing address verification (the **-bv** option) and testing incoming message policy controls (the **-bh** option).
- For a daemon, queue runner, delivery, or address testing process, the uid remains as root at this stage, but the gid is changed to the Exim group.

The processes that initially retain root privilege behave as follows:

- A daemon process changes the uid to the Exim user after setting up one or more listening sockets.
- A queue runner process retains root privilege throughout its execution. Its job is to fork a controlled sequence of delivery processes.
- A delivery process retains root privilege throughout most of its execution, but any actual deliveries (that is, the transports themselves) are run in subprocesses which always change to a non-root uid and gid. For local deliveries this is typically the uid and gid of the owner of the mailbox; for remote deliveries, the Exim uid and gid are used. Once all the delivery subprocesses have been run, a delivery process changes to the Exim uid and gid while doing post-delivery tidying up such as updating the retry database and generating bounce and warning messages.

  While the recipient addresses in a message are being routed, the delivery process runs as root. However, if a user's filter file has to be processed, this is done in a subprocess that runs under the individual user's uid and gid. A system filter is run as root unless **system_filter_user** is set.
- A process that is testing addresses (the **-bt** option) runs as root so that the routing is done in the same environment as a message delivery.

47.2 Running Exim without privilege

Some installations like to run Exim in an unprivileged state for more of its operation, for added security. Support for this mode of operation is provided by the global option **deliver_drop_privilege**. When this is set, the uid and gid are changed to the Exim user and group at the start of a delivery process (and also queue runner and address testing processes). This means that address routing is no longer run as root, and the deliveries themselves cannot change to any other uid.

Leaving the binary setuid to root, but setting **deliver_drop_privilege** means that the daemon can still be started in the usual way, and it can respond correctly to SIGHUP because the re-invocation regains root privilege.

An alternative approach is to make Exim setuid/setgid to the Exim user and group. If you do this, the daemon must be started from a root process. (Calling Exim from a root process makes it behave in the way it does when it is setuid root.) However, the daemon cannot restart itself after a SIGHUP signal because it cannot regain privilege.

It is still useful to set **deliver_drop_privilege** in this case, because it stops Exim from trying to re-invoke itself to do a delivery after a message has been received. Such a re-invocation is a waste of resources because it has no effect.

If restarting the daemon is not an issue (for example, if *inetd* is being used instead of a daemon), having the binary setuid to the Exim user seems a clean approach, but there is one complication:

In this style of operation, Exim is running with the real uid and gid set to those of the calling process, and the effective uid/gid set to Exim's values. Ideally, any association with the calling process' uid/gid should be dropped, that is, the real uid/gid should be reset to the effective values so as to discard any privileges that the caller may have. While some operating systems have a function that permits this action for a non-root effective uid, quite a number of them do not. Because of this lack of standardization, Exim does not address this problem at this time.

For this reason, the recommended approach for 'mostly unprivileged' running is to keep the Exim binary setuid to root, and to set **deliver_drop_privilege**. This also has the advantage of allowing a daemon to be used in the most straightforward way.

If you configure Exim not to run delivery processes as root, there are a number of restrictions on what you can do:

- You can deliver only as the Exim user/group. You should explicitly use the **user** and **group** options to override routers or local transports that normally deliver as the recipient. This makes sure that configurations that work in this mode function the same way in normal mode. Any implicit or explicit specification of another user causes an error.

- Use of **.forward** files is severely restricted, such that it is usually not worthwhile to include them in the configuration.
- Users who wish to use **.forward** would have to make their home directory and the file itself accessible to the Exim user. Pipe and append-to-file entries, and their equivalents in Exim filters, cannot be used. While they could be enabled in the Exim user's name, that would be insecure and not very useful.
- Unless the local user mailboxes are all owned by the Exim user (possible in some POP3 or IMAP-only environments):
    - They must be owned by the Exim group and be writable by that group. This implies you must set **mode** in the appendfile configuration, as well as the mode of the mailbox files themselves.
    - You must set **no_check_owner**, since most or all of the files will not be owned by the Exim user.
    - You must set **file_must_exist**, because Exim cannot set the owner correctly on a newly created mailbox when unprivileged. This also implies that new mailboxes need to be created manually.

These restrictions severely restrict what can be done in local deliveries. However, there are no restrictions on remote deliveries. If you are running a gateway host that does no local deliveries, setting **deliver_drop_privilege** gives more security at essentially no cost.

47.3 Delivering to local files

Full details of the checks applied by **appendfile** before it writes to a file are given in chapter 25.

47.4 IPv4 source routing

Many operating systems suppress IP source-routed packets in the kernel, but some cannot be made to do this, so Exim does its own check. It logs incoming IPv4 source-routed TCP calls, and then drops them. Things are all different in IPv6. No special checking is currently done.

47.5 The VRFY, EXPN, and ETRN commands in SMTP

Support for these SMTP commands is disabled by default. If required, they can be enabled by defining suitable ACLs.

47.6 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. Untrusted users may also be permitted to use specific forms of address with the **-f** option by setting the **untrusted_set_sender** option.

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, 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. This restriction can be relaxed by setting the **no_prod_requires_admin** option. Similarly, the use of **-bp** (and its variants) to list the contents of the queue is also restricted to admin users. This restriction can be relaxed by setting **no_queue_list_requires_admin**.

Exim recognises an admin user if the calling process is running as root or as the Exim user or if any of the groups associated with the calling process is the Exim group. It is not necessary actually to be running under the Exim group. However, if admin users who are not root or the Exim user 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.

47.7 Spool files

Exim's spool directory and everything it contains is owned by the Exim user and set to the Exim group. The mode for spool files is defined in the **Local/Makefile** configuration file, and defaults to 0640. This means that any user who is a member of the Exim group can access these files.

47.8 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.

47.9 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.

47.10 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.

47.11 Use of sprintf()

A large number of occurrences of 'sprintf' in the code are actually calls to *string_sprintf()*, a function that 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.

The remaining uses of *sprintf()* happen in controlled circumstances where the output buffer is known to be sufficiently long to contain the converted string.

47.12 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.

47.13 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.

48. Format of spool files

A message on Exim's queue 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 disk crashes where the directory is lost but the files themselves are recoverable.

Files whose names end with -J may also be seen in the **input** directory (or its subdirectories when **split_spool_directory** is set). 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 -H file contains the login name for the uid of the process that called Exim to read the message, followed by the numerical uid and gid. For a locally generated message, this is normally the user who sent the message. For a message received over TCP/IP, it is normally the Exim user.

The third line of the file contains the address of the message's sender as transmitted in the envelope, contained in angle brackets. The sender address is empty for bounce messages. For incoming SMTP mail, the sender address is given in the MAIL command. For locally generated mail, the sender address is created by Exim from the login name of the current user and the configured **qualify_domain**. However, this can be overridden by the **-f** option or a leading 'From' line if the caller is trusted, or if the supplied address is '<>' or an address that matches **untrusted_set_senders**.

The fourth line contains two numbers. The first is the time that the message was received, in the conventional Unix form – the 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 lines starting with a hyphen. These can appear in any order, and are omitted when not relevant:

- **-auth_id** <*text*>: The id information for a message received on an authenticated SMTP connection – the value of the **$authenticated_id** variable.
- **-auth_sender** <*address*>: The address of an authenticated sender – the value of the **$authenticated_sender** variable.
- **-body_linecount** <*number*>: This records the number of lines in the body of the message, and is always present.
- **-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*>.
- **-helo_name** <*text*>: This records the host name as specified by a remote host in a HELO or EHLO command.
- **-host_address** <*address*>**.**<*port*>: This records the IP address of the host from which the message was received and the remote port number that was used. It is omitted for locally generated messages.
- **-host_auth** <*text*>: If the message was received on an authenticated SMTP connection, this records the name of the authenticator – the value of the **$sender_host_authenticated** variable.
- **-host_lookup_failed**: This is present if an attempt to look up the sending host's name from its IP address failed. It corresponds to the **$host_lookup_failed** variable.
- **-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 when the message was being received. It is not present if no reverse lookup was done.

- **-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, if any.
- **-interface_address** <*address*>**.**<*port*>: This records the IP address of the local interface and the port number through which a message was received from a remote host. It is omitted for locally generated messages.
- **-local**: The message is from a local sender.
- **-localerror**: The message is a locally-generated bounce message.
- **-local_scan** <*string*>: This records the data string that was returned by the *local_scan()* function when the message was received – the value of the **$local_scan_data** variable. It is omitted if no data was returned.
- **-manual_thaw**: The message was frozen but has been thawed manually, that is, by an explicit Exim command rather than via the auto-thaw process.
- **-N**: A testing delivery process was started using the **-N** option to suppress any actual deliveries, but delivery was deferred. At any further delivery attempts, **-N** is assumed.
- **-received_protocol**: This records the value of the **$received_protocol** variable, which contains the name of the protocol by which the message was received.
- **-sender_set_untrusted**: The envelope sender of this message was set by an untrusted local caller (used to ensure that the caller is displayed in queue listings).
- **-tls_certificate_verified**: A TLS certificate was received from the client that sent this message, and the certificate was verified by the server.
- **-tls_cipher** <*cipher name*>: When the message was received over an encrypted connection, this records the name of the cipher suite that was used.
- **-tls_peerdn** <*peer DN*>: When the message was received over an encrypted connection, and a certificate was received from the client, this records the Distinguished Name from that certificate.

Following the options there is a list of 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 complete.

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.example
NN alice@wonderland.fict.example
NN editor@thesaurus.ref.example
```

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.example
darcy@austen.fict.example
rdo@foundation
alice@wonderland.fict.example
```

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 a **redirect** router, each line is of the following form:

<top-level address> <errors_to address> <length>,<parent number>#<flag bits>

The 01 flag bit indicates the presence of the three other fields that follow the top-level address. Other bits may be used in future to support additional fields. The *<parent number>* is the offset in the recipients list of the original parent of the ‘one time’ address. The first two fields are the envelope sender that is associated with this address and its length. If the length is zero, there is no special envelope sender (there are then two space characters in the line). A non-empty field can arise from a **redirect** router that has an **errors_to** setting.

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. Here is a typical set of headers:

```
111P Received: by hobbit.fict.example with local (Exim 4.00)
        id 14y9EI-00026G-00; Fri, 11 May 2001 10:28:59 +0100
049  Message-Id: <E14y9EI-00026G-00@hobbit.fict.example>
038* X-rewrote-sender: bb@hobbit.fict.example
042* From: Bilbo Baggins <bb@hobbit.fict.example>
049F From: Bilbo Baggins <B.Baggins@hobbit.fict.example>
099* To: alice@wonderland.fict.example, rdo@foundation,
 darcy@austen.fict.example, editor@thesaurus.ref.example
109T To: alice@wonderland.fict.example, rdo@foundation.fict.example,
 darcy@austen.fict.example, editor@thesaurus.ref.example
038  Date: Fri, 11 May 2001 10:28:59 +0100
```

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*.

49. Adding new drivers or lookup types

The following actions have to be taken in order to add a new router, transport, authenticator, 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 ROUTER, TRANSPORT, AUTH, 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/routers**, **src/transports**, **src/auths**, 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