SENDMAIL

INSTALLATION AND OPERATION GUIDE

Eric Allman
Pangæa Reference Systems
eric@Sendmail.ORG

Version 8.70

For Sendmail Version 8.7

Sendmail implements a general purpose internetwork mail routing facility under the UNIX® operating system. It is not tied to any one transport protocol — its function may be likened to a crossbar switch, relaying messages from one domain into another. In the process, it can do a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for *sendmail*, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting an existing configuration files incrementally.

Sendmail is based on RFC821 (Simple Mail Transport Protocol), RFC822 (Internet Mail Format Protocol), RFC1123 (Internet Host Requirements), RFC1521 (MIME), RFC1651 (SMTP Service Extensions), and a series of as-yet-draft standards describing Delivery Status Notifications (DSNs), available from the internet drafts sites as draft-ietf-notary-mime-delivery-*XX*.txt, draft-ietf-notary-mime-report-*XX*.txt, draft-ietf-notary-smtp-drpt-*XX*.txt, and draft-ietf-notary-status-*XX*.txt (replace *XX* by the latest draft number). However, since *sendmail* is designed to work in a wider world, in many cases it can be configured to exceed these protocols. These cases are described herein.

Although *sendmail* is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances. These features are described.

Section one describes how to do a basic *sendmail* installation. Section two explains the day-to-day information you should know to maintain your mail system. If you have a relatively normal site, these two sections should contain sufficient information for you to install *sendmail* and keep it happy. Section three describes some parameters that may be safely tweaked. Section four has information regarding the command line arguments. Section five contains the nitty-gritty information about the configuration file. This section is for masochists and people who must write their own configuration file. Section six describes configuration that can be done at compile time. Section seven gives a brief description of differences in this version of *sendmail*. The appendixes give a brief but detailed explanation of a number of features not described in the rest of the paper.

WARNING: Several major changes were introduced in version 8.7. You should not attempt to use this document for prior versions of *sendmail*.

1. BASIC INSTALLATION

There are two basic steps to installing *sendmail*. The hard part is to build the configuration table. This is a file that *sendmail* reads when it starts up that describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although the configuration table is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second part is actually doing the installation, i.e., creating the necessary files, etc.

The remainder of this section will describe the installation of *sendmail* assuming you can use one of the existing configurations and that the standard installation parameters are acceptable. All pathnames and examples are given from the root of the *sendmail* subtree, normally */usr/src/usr.sbin/sendmail* on 4.4BSD.

If you are loading this off the tape, continue with the next section. If you have a running binary already on your system, you should probably skip to section 1.2.

1.1. Compiling Sendmail

All *sendmail* source is in the *src* subdirectory. If you are running on a 4.4BSD system, compile by typing ''make''. On other systems, you may have to make some other adjustments. On most systems, you can do the appropriate compilation by typing

```
sh makesendmail
```

This will leave the binary in an appropriately named subdirectory. It works for multiple object versions compiled out of the same directory.

1.1.1. Tweaking the Makefile

Sendmail supports two different formats for the local (on disk) version of databases, notably the *aliases* database. At least one of these should be defined if at all possible.

NDBM — The ''new DBM'' format, available on nearly all systems around today. This was the preferred format prior to 4.4BSD. It allows such complex things as multiple databases and closing a currently open database.

NEWDB — The new database package from Berkeley. If you have this, use it. It allows long records, multiple open databases, real in-memory caching, and so forth. You can define this in conjunction with one of the other two; if you do, old databases are read, but when a new database is created it will be in NEWDB format. As a nasty hack, if you have NEWDB, NDBM, and NIS defined, and if the alias file name includes the substring ''/yp/'', *sendmail* will create both new and old versions of the alias file during a *newalias* command. This is required because the Sun NIS/YP system reads the DBM version of the alias file. It's ugly as sin, but it works.

If neither of these are defined, *sendmail* reads the alias file into memory on every invocation. This can be slow and should be avoided. There are also several methods for remote database access:

NIS — Sun's Network Information Services (formerly YP).

NISPLUS — Sun's NIS+ services.

NETINFO — NeXT's NetInfo service.

HESIOD — Hesiod service (from Athena).

Other compilation flags are set in conf.h and should be predefined for you unless you are porting to a new environment.

1.1.2. Compilation and installation

After making the local system configuration described above, You should be able to compile and install the system. The script ‘‘makesendmail’’ is the best approach on most systems:

sh makesendmail

This will use *uname*(1) to select the correct Makefile for your environment.

You may be able to install using

sh makesendmail install

This should install the binary in /usr/sbin and create links from /usr/bin/newaliases and /usr/bin/mailq to /usr/sbin/sendmail. On 4.4BSD systems it will also format and install man pages.

1.2. Configuration Files

Sendmail cannot operate without a configuration file. The configuration defines the mail delivery mechanisms understood at this site, how to access them, how to forward email to remote mail systems, and a number of tuning parameters. This configuration file is detailed in the later portion of this document.

The *sendmail* configuration can be daunting at first. The world is complex, and the mail configuration reflects that. The distribution includes an m4-based configuration package that hides a lot of the complexity.

These configuration files are simpler than old versions largely because the world has become simpler; in particular, text-based host files are officially eliminated, obviating the need to ‘‘hide’’ hosts behind a registered internet gateway.

These files also assume that most of your neighbors use domain-based UUCP addressing; that is, instead of naming hosts as ‘‘host!user’’ they will use ‘‘host.domain!user’’. The configuration files can be customized to work around this, but it is more complex.

Our configuration files are processed by *m4* to facilitate local customization; the directory *cf* of the *sendmail* distribution directory contains the source files. This directory contains several subdirectories:

cf — Both site-dependent and site-independent descriptions of hosts. These can be literal host names (e.g., ‘‘ucbvax.mc’’) when the hosts are gateways or more general descriptions (such as ‘‘tcpproto.mc’’ as a general description of an SMTP-connected host or ‘‘uucpproto.mc’’ as a general description of a UUCP-connected host). Files ending **.mc** (‘‘Master Configuration’’) are the input descriptions; the output is in the corresponding **.cf** file. The general structure of these files is described below.

domain — Site-dependent subdomain descriptions. These are tied to the way your organization wants to do addressing. For example, **domain/cs.exposed.m4** is our description for hosts in the CS.Berkeley.EDU subdomain that want their individual hostname to be externally visible; **domain/cs.hidden.m4** is the same except that the hostname is hidden (everything looks like it comes from CS.Berkeley.EDU). These are referenced using the DOMAIN **m4** macro in the **.mc** file.

feature — Definitions of specific features that some particular host in your site might want. These are referenced using the FEATURE **m4** macro. An example feature is use_cw_file (which tells *sendmail* to read an /etc/sendmail.cw file on startup to find the set of local names).

hack — Local hacks, referenced using the HACK **m4** macro. Try to avoid these. The point of having them here is to make it clear that they smell.

m4 — Site-independent *m4*(1) include files that have information common to all configuration files. This can be thought of as a ‘‘#include’’ directory.

mailer — Definitions of mailers, referenced using the MAILER **m4** macro. The mailer types that are known in this distribution are fax, local, smtp, uucp, and usenet. For example, to include support for the UUCP-based mailers, use ‘‘MAILER(uucp)’’.

ostype — Definitions describing various operating system environments (such as the location of support files). These are referenced using the OSTYPE **m4** macro.

sh — Shell files used by the **m4** build process. You shouldn't have to mess with these.

siteconfig — Local site configuration information, such as UUCP connectivity. They normally contain lists of site information, for example:

```
SITE(contessa)
SITE(hoptoad)
SITE(nkainc)
SITE(well)
```

They are referenced using the SITECONFIG macro:

```
SITECONFIG(site.config.file, name_of_site, X)
```

where *X* is the macro/class name to use. It can be U (indicating locally connected hosts) or one of W, X, or Y for up to three remote UUCP hubs.

If you are in a new domain (e.g., a company), you will probably want to create a cf/domain file for your domain. This consists primarily of relay definitions: for example, Berkeley's domain definition defines relays for BitNET, CSNET, and UUCP. Of these, only the UUCP relay is particularly specific to Berkeley. All of these are internet-style domain names. Please check to make certain they are reasonable for your domain.

Subdomains at Berkeley are also represented in the cf/domain directory. For example, the domain cs-exposed is the Computer Science subdomain with the local hostname shown to other users; cs-hidden makes users appear to be from the CS.Berkeley.EDU subdomain (with no local host information included). You will probably have to update this directory to be appropriate for your domain.

You will have to use or create **.mc** files in the *cf/cf* subdirectory for your hosts. This is detailed in the cf/README file.

1.3. Details of Installation Files

This subsection describes the files that comprise the *sendmail* installation.

1.3.1. /usr/sbin/sendmail

The binary for *sendmail* is located in /usr/sbin[1]. It should be setuid root. For security reasons, /, /usr, and /usr/sbin should be owned by root, mode 755[2].

[1]This is usually /usr/sbin on 4.4BSD and newer systems; many systems install it in /usr/lib. I understand it is in /usr/ucblib on System V Release 4.

[2]Some vendors ship them owned by bin; this creates a security hole that is not actually related to *sendmail*. Other important directories that should have restrictive ownerships and permissions are /bin, /usr/bin, /etc, /usr/etc, /lib, and /usr/lib.

1.3.2. /etc/sendmail.cf

This is the configuration file for *sendmail*[3]. This and /etc/sendmail.pid are the only non-library file names compiled into *sendmail*[4].

The configuration file is normally created using the distribution files described above. If you have a particularly unusual system configuration you may need to create a special version. The format of this file is detailed in later sections of this document.

1.3.3. /usr/bin/newaliases

The *newaliases* command should just be a link to *sendmail*:

```
rm –f /usr/bin/newaliases
ln –s /usr/sbin/sendmail /usr/bin/newaliases
```

This can be installed in whatever search path you prefer for your system.

1.3.4. /var/spool/mqueue

The directory */var/spool/mqueue* should be created to hold the mail queue. This directory should be mode 700 and owned by root.

The actual path of this directory is defined in the **Q** option of the *sendmail.cf* file.

1.3.5. /etc/aliases*

The system aliases are held in ‘‘/etc/aliases’’. A sample is given in ‘‘lib/aliases’’ which includes some aliases which *must* be defined:

```
cp lib/aliases /etc/aliases
edit /etc/aliases
```

You should extend this file with any aliases that are apropos to your system.

Normally *sendmail* looks at a version of these files maintained by the *dbm*(3) or *db*(3) routines. These are stored either in ‘‘/etc/aliases.dir’’ and ‘‘/etc/aliases.pag’’ or ‘‘/etc/aliases.db’’ depending on which database package you are using. These can initially be created as empty files, but they will have to be initialized promptly. These should be mode 644:

```
cp /dev/null /etc/aliases.dir
cp /dev/null /etc/aliases.pag
chmod 644 /etc/aliases.*
newaliases
```

The *db* routines preset the mode reasonably, so this step can be skipped. The actual path of this file is defined in the **A** option of the *sendmail.cf* file.

1.3.6. /etc/rc

It will be necessary to start up the *sendmail* daemon when your system reboots. This daemon performs two functions: it listens on the SMTP socket for connections (to receive mail from a remote system) and it processes the queue periodically to insure that mail gets delivered when hosts come up.

[3]Actually, the pathname varies depending on the operating system; /etc is the preferred directory. Some older systems install it in **/usr/lib/sendmail.cf**, and I’ve also seen it in **/usr/ucblib** and **/etc/mail**. If you want to move this file, change *src/conf.h*.

[4]The system libraries can reference other files; in particular, system library subroutines that *sendmail* calls probably reference */etc/passwd* and */etc/resolv.conf*.

Add the following lines to ‘‘/etc/rc’’ (or ‘‘/etc/rc.local’’ as appropriate) in the area where it is starting up the daemons:

```
if [ –f /usr/sbin/sendmail –a –f /etc/sendmail.cf ]; then
        (cd /var/spool/mqueue; rm –f [lnx]f*)
        /usr/sbin/sendmail –bd –q30m &
        echo –n ’ sendmail’ >/dev/console
fi
```

The ‘‘cd’’ and ‘‘rm’’ commands insure that all lock files have been removed; extraneous lock files may be left around if the system goes down in the middle of processing a message. The line that actually invokes *sendmail* has two flags: ‘‘–bd’’ causes it to listen on the SMTP port, and ‘‘–q30m’’ causes it to run the queue every half hour.

Some people use a more complex startup script, removing zero length qf files and df files for which there is no qf file. For example, see Figure 1 for an example of a complex startup script.

If you are not running a version of UNIX that supports Berkeley TCP/IP, do not include the **–bd** flag.

1.3.7. /usr/lib/sendmail.hf

This is the help file used by the SMTP **HELP** command. It should be copied from ‘‘lib/sendmail.hf’’:

```
cp lib/sendmail.hf /usr/lib
```

The actual path of this file is defined in the **H** option of the *sendmail.cf* file.

1.3.8. /etc/sendmail.st

If you wish to collect statistics about your mail traffic, you should create the file ‘‘/etc/sendmail.st’’:

```
cp /dev/null /etc/sendmail.st
chmod 666 /etc/sendmail.st
```

This file does not grow. It is printed with the program ‘‘mailstats/mailstats.c.’’ The actual path of this file is defined in the **S** option of the *sendmail.cf* file.

1.3.9. /usr/bin/mailq

If *sendmail* is invoked as ‘‘mailq,’’ it will simulate the **–bp** flag (i.e., *sendmail* will print the contents of the mail queue; see below). This should be a link to /usr/sbin/sendmail.

2. NORMAL OPERATIONS

2.1. The System Log

The system log is supported by the *syslogd*(8) program. All messages from *sendmail* are logged under the LOG_MAIL facility[5].

2.1.1. Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the local area network), the word ‘‘send-

[5]Except on Ultrix, which does not support facilities in the syslog.

```
# remove zero length qf files
for qffile in qf*
do
        if [ –r $qffile ]
        then
                if [ ! –s $qffile ]
                then
                        echo –n " <zero: $qffile>" > /dev/console
                        rm –f $qffile
                fi
        fi
done
# rename tf files to be qf if the qf does not exist
for tffile in tf*
do
        qffile=‘echo $tffile | sed ’s/t/q/’‘
        if [ –r $tffile –a ! –f $qffile ]
        then
                echo –n " <recovering: $tffile>" > /dev/console
                mv $tffile $qffile
        else
                echo –n " <extra: $tffile>" > /dev/console
                rm –f $tffile
        fi
done
# remove df files with no corresponding qf files
for dffile in df*
do
        qffile=‘echo $dffile | sed ’s/d/q/’‘
        if [ –r $dffile –a ! –f $qffile ]
        then
                echo –n " <incomplete: $dffile>" > /dev/console
                mv $dffile ‘echo $dffile | sed ’s/d/D/’‘
        fi
done
# announce files that have been saved during disaster recovery
for xffile in [A-Z]f*
do
        echo –n " <panic: $xffile>" > /dev/console
done
```

Figure 1 — A complex startup script

mail:’’, and a message[6]. Most messages are a sequence of *name=value* pairs.

[6]This format may vary slightly if your vendor has changed the syntax.

The two most common lines are logged when a message is processed. The first logs the receipt of a message; there will be exactly one of these per message. Some fields may be omitted if they do not contain interesting information. Fields are:

from	The envelope sender address.
size	The size of the message in bytes.
class	The class (i.e., numeric precedence) of the message.
pri	The initial message priority (used for queue sorting).
nrcpts	The number of envelope recipients for this message (after aliasing and forwarding).
msgid	The message id of the message (from the header).
proto	The protocol used to receive this message (e.g., ESMTP or UUCP)
relay	The machine from which it was received.

There is also one line logged per delivery attempt (so there can be several per message if delivery is deferred or there are multiple recipients). Fields are:

to	A comma-separated list of the recipients to this mailer.
ctladdr	The ‘‘controlling user’’, that is, the name of the user whose credentials we use for delivery.
delay	The total delay between the time this message was received and the time it was delivered.
xdelay	The amount of time needed in this delivery attempt (normally indicative of the speed of the connection).
mailer	The name of the mailer used to deliver to this recipient.
relay	The name of the host that actually accepted (or rejected) this recipient.
stat	The delivery status.

Not all fields are present in all messages; for example, the relay is not listed for local deliveries.

2.1.2. Levels

If you have *syslogd*(8) or an equivalent installed, you will be able to do logging. There is a large amount of information that can be logged. The log is arranged as a succession of levels. At the lowest level only extremely strange situations are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under ten are considered generally ‘‘useful;’’ log levels above 64 are reserved for debugging purposes. Levels from 11–64 are reserved for verbose information that some sites might want.

A complete description of the log levels is given in section 4.6.

2.2. Dumping State

You can ask *sendmail* to log a dump of the open files and the connection cache by sending it a SIGUSR1 signal. The results are logged at LOG_DEBUG priority.

2.3. The Mail Queue

Sometimes a host cannot handle a message immediately. For example, it may be down or overloaded, causing it to refuse connections. The sending host is then expected to save this message in its mail queue and attempt to deliver it later.

Under normal conditions the mail queue will be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although *sendmail* ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

2.3.1. Printing the queue

The contents of the queue can be printed using the *mailq* command (or by specifying the **–bp** flag to *sendmail*):

```
mailq
```

This will produce a listing of the queue id's, the size of the message, the date the message entered the queue, and the sender and recipients.

2.3.2. Forcing the queue

Sendmail should run the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, *sendmail* first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to insure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process (however, *sendmail* does include heuristics to try to abort jobs that are taking absurd amounts of time; technically, this violates RFC 821, but is blessed by RFC 1123). Due to the locking algorithm, it is impossible for one job to freeze the entire queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no completely general way to solve this.

In some cases, you may find that a major host going down for a couple of days may create a prohibitively large queue. This will result in *sendmail* spending an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
cd /var/spool
mv mqueue omqueue; mkdir mqueue; chmod 700 mqueue
```

You should then kill the existing daemon (since it will still be processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

```
/usr/sbin/sendmail –oQ/var/spool/omqueue –q
```

The **–oQ** flag specifies an alternate queue directory and the **–q** flag says to just run every job in the queue. If you have a tendency toward voyeurism, you can use the **–v** flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
rmdir /var/spool/omqueue
```

2.4. The Service Switch

The implementation of certain system services such as host and user name lookup is controlled by the service switch. If the host operating system supports such a switch *sendmail* will use the native version. Ultrix, Solaris, and DEC OSF/1 are examples of such systems.

If the underlying operating system does not support a service switch (e.g., SunOS, HP-UX, BSD) then *sendmail* will provide a stub implementation. The **ServiceSwitchFile** option points to

the name of a file that has the service definitions Each line has the name of a service and the possible implementations of that service. For example, the file:

```
hosts    dns files nis
aliases  files nis
```

will ask *sendmail* to look for hosts in the Domain Name System first. If the requested host name is not found, it tries local files, and if that fails it tries NIS. Similarly, when looking for aliases it will try the local files first followed by NIS.

Service switches are not completely integrated. For example, despite the fact that the host entry listed in the above example specifies to look in NIS, on SunOS this won't happen because the system implementation of *gethostbyname*(3) doesn't understand this. If there is enough demand *sendmail* may reimplement *gethostbyname*(3), *gethostbyaddr*(3), *getpwent*(3), and the other system routines that would be necessary to make this work seamlessly.

2.5. The Alias Database

The alias database exists in two forms. One is a text form, maintained in the file */etc/aliases*. The aliases are of the form

```
name: name1, name2, ...
```

Only local names may be aliased; e.g.,

```
eric@prep.ai.MIT.EDU: eric@CS.Berkeley.EDU
```

will not have the desired effect (except on prep.ai.MIT.EDU, and they probably don't want me)[7]. Aliases may be continued by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a sharp sign (''#'') are comments.

The second form is processed by the *ndbm*(3)[8] or *db*(3) library. This form is in the files */etc/aliases.dir* and */etc/aliases.pag*. This is the form that *sendmail* actually uses to resolve aliases. This technique is used to improve performance.

The control of search order is actually set by the service switch. Essentially, the entry

```
OAswitch:aliases
```

is always added as the first alias entry; also, the first alias file name without a class (e.g., without ''nis:'' on the front) will be used as the name of the file for a ''files'' entry in the aliases switch. For example, if the configuration file contains

```
OA/etc/aliases
```

and the service switch contains

```
aliases  nis files nisplus
```

then aliases will first be searched in the NIS database, then in /etc/aliases, then in the NIS+ database.

You can also use NIS-based alias files. For example, the specification:

```
OA/etc/aliases
OAnis:mail.aliases@my.nis.domain
```

will first search the /etc/aliases file and then the map named ''mail.aliases'' in ''my.nis.domain''. Warning: if you build your own NIS-based alias files, be sure to provide the **–l** flag to *makedbm*(8) to map upper case letters in the keys to lower case; otherwise, aliases with upper case letters in

[7]Actually, any mailer that has the 'A' mailer flag set will permit aliasing; this is normally limited to the local mailer.

[8]The *gdbm* package probably works as well.

their names won't match incoming addresses.

Additional flags can be added after the colon exactly like a **K** line — for example:

```
OAnis:–N mail.aliases@my.nis.domain
```

will search the appropriate NIS map and always include null bytes in the key.

2.5.1. Rebuilding the alias database

The DB or DBM version of the database may be rebuilt explicitly by executing the command

```
newaliases
```

This is equivalent to giving *sendmail* the **–bi** flag:

```
/usr/sbin/sendmail –bi
```

If the **RebuildAliases** (old **D**) option is specified in the configuration, *sendmail* will rebuild the alias database automatically if possible when it is out of date. Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than the rebuild timeout (option **AliasWait**, old **a**, which is normally five minutes) to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

If you have multiple aliases databases specified, the **–bi** flag rebuilds all the database types it understands (for example, it can rebuild NDBM databases but not NIS databases).

2.5.2. Potential problems

There are a number of problems that can occur with the alias database. They all result from a *sendmail* process accessing the DBM version while it is only partially built. This can happen under two circumstances: One process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

Sendmail has three techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone aborting the process leaving a partially rebuilt database. Second, it locks the database source file during the rebuild — but that may not work over NFS or if the file is unwritable. Third, at the end of the rebuild it adds an alias of the form

```
@: @
```

(which is not normally legal). Before *sendmail* will access the database, it checks to insure that this entry exists[9].

2.5.3. List owners

If an error occurs on sending to a certain address, say ''*x*'', *sendmail* will look for an alias of the form ''owner-*x*'' to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintenance of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
        sam@matisse
owner-unix-wizards: unix-wizards-request
unix-wizards-request: eric@ucbarpa
```

[9]The **AliasWait** option is required in the configuration for this action to occur. This should normally be specified.

would cause ‘‘eric@ucbarpa’’ to get the error that will occur when someone sends to unix-wizards due to the inclusion of ‘‘nosuchuser’’ on the list.

List owners also cause the envelope sender address to be modified. The contents of the owner alias are used if they point to a single user, otherwise the name of the alias itself is used. For this reason, and to obey Internet conventions, the ‘‘owner-’’ address normally points at the ‘‘-request’’ address; this causes messages to go out with the typical Internet convention of using ‘‘*list*-request’’ as the return address.

2.6. User Information Database

If you have a version of *sendmail* with the user information database compiled in, and you have specified one or more databases using the **U** option, the databases will be searched for a *user*:maildrop entry. If found, the mail will be sent to the specified address.

2.7. Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user may put a file with the name ‘‘.forward’’ in his or her home directory. If this file exists, *sendmail* redirects mail for that user to the list of addresses listed in the .forward file. For example, if the home directory for user ‘‘mckusick’’ has a .forward file with contents:

```
mckusick@ernie
kirk@calder
```

then any mail arriving for ‘‘mckusick’’ will be redirected to the specified accounts.

Actually, the configuration file defines a sequence of filenames to check. By default, this is the user’s .forward file, but can be defined to be more generally using the **J** option. If you change this, you will have to inform your user base of the change; .forward is pretty well incorporated into the collective subconscious.

2.8. Special Header Lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into *sendmail* that cannot be changed without changing the code. These builtins are described here.

2.8.1. Errors-To:

If errors occur anywhere during processing, this header will cause error messages to go to the listed addresses. This is intended for mailing lists.

The Errors-To: header was created in the bad old days when UUCP didn’t understand the distinction between an envelope and a header; this was a hack to provide what should now be passed as the envelope sender address. It should go away. It is only used if the **UseErrorsTo** option is set.

The Errors-To: header is official deprecated and will go away in a future release.

2.8.2. Apparently-To:

RFC 822 requires at least one recipient field (To:, Cc:, or Bcc: line) in every message. If a message comes in with no recipients listed in the message then *sendmail* will adjust the header based on the ‘‘NoRecipientAction’’ option. One of the possible actions is to add an ‘‘Apparently-To:’’ header line for any recipients it is aware of. This is not put in as a standard recipient line to warn any recipients that the list is not complete.

The Apparently-To: header is non-standard and is deprecated.

2.8.3. Precedence

The Precedence: header can be used as a crude control of message priority. It tweaks the sort order in the queue and can be configured to change the message timeout values.

2.9. IDENT Protocol Support

Sendmail supports the IDENT protocol as defined in RFC 1413. Although this enhances identification of the author of an email message by doing a ‘‘call back’’ to the originating system to include the owner of a particular TCP connection in the audit trail it is in no sense perfect; a determined forger can easily spoof the IDENT protocol. The following description is excerpted from RFC 1413:

> 6. Security Considerations
>
> The information returned by this protocol is at most as trustworthy as the host providing it OR the organization operating the host. For example, a PC in an open lab has few if any controls on it to prevent a user from having this protocol return any identifier the user wants. Likewise, if the host has been compromised the information returned may be completely erroneous and misleading.
>
> The Identification Protocol is not intended as an authorization or access control protocol. At best, it provides some additional auditing information with respect to TCP connections. At worst, it can provide misleading, incorrect, or maliciously incorrect information.
>
> The use of the information returned by this protocol for other than auditing is strongly discouraged. Specifically, using Identification Protocol information to make access control decisions - either as the primary method (i.e., no other checks) or as an adjunct to other methods may result in a weakening of normal host security.
>
> An Identification server may reveal information about users, entities, objects or processes which might normally be considered private. An Identification server provides service which is a rough analog of the CallerID services provided by some phone companies and many of the same privacy considerations and arguments that apply to the CallerID service apply to Identification. If you wouldn’t run a "finger" server due to privacy considerations you may not want to run this protocol.

In some cases your system may not work properly with IDENT support due to a bug in the TCP/IP implementation. The symptoms will be that for some hosts the SMTP connection will be closed almost immediately. If this is true or if you do not want to use IDENT, you should set the IDENT timeout to zero; this will disable the IDENT protocol.

3. ARGUMENTS

The complete list of arguments to *sendmail* is described in detail in Appendix A. Some important arguments are described here.

3.1. Queue Interval

The amount of time between forking a process to run through the queue is defined by the **–q** flag. If you run with delivery mode set to **i** or **b** this can be relatively large, since it will only be relevant when a host that was down comes back up. If you run in **q** mode it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue. (See also the MinQueueAge option.)

RFC 1123 section 5.3.1.1 says that this value should be at least 30 minutes (although that probably doesn’t make sense if you use ‘‘queue-only’’ mode).

3.2. Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the **–bd** flag. The **–bd** flag and the **–q** flag may be combined in one call:

```
/usr/sbin/sendmail –bd –q30m
```

An alternative approach is to invoke sendmail from *inetd*(8) (use the **–bs** flag to ask sendmail to speak SMTP on its standard input and output). This works and allows you to wrap *sendmail* in a TCP wrapper program, but may be a bit slower since the configuration file has to be reread on every message that comes in. If you do this, you still need to have a *sendmail* running to flush the queue:

```
/usr/sbin/sendmail –q30m
```

3.3. Forcing the Queue

In some cases you may find that the queue has gotten clogged for some reason. You can force a queue run using the **–q** flag (with no value). It is entertaining to use the **–v** flag (verbose) when this is done to watch what happens:

```
/usr/sbin/sendmail –q –v
```

You can also limit the jobs to those with a particular queue identifier, sender, or recipient using one of the queue modifiers. For example, ‘‘–qRberkeley’’ restricts the queue run to jobs that have the string ‘‘berkeley’’ somewhere in one of the recipient addresses. Similarly, ‘‘–qSstring’’ limits the run to particular senders and ‘‘–qIstring’’ limits it to particular queue identifiers.

3.4. Debugging

There are a fairly large number of debug flags built into *sendmail*. Each debug flag has a number and a level, where higher levels means to print out more information. The convention is that levels greater than nine are ‘‘absurd,’’ i.e., they print out so much information that you wouldn’t normally want to see them except for debugging that particular piece of code. Debug flags are set using the **–d** option; the syntax is:

```
debug-flag:    –d debug-list
debug-list:    debug-option [ , debug-option ]*
debug-option:  debug-range [ . debug-level ]
debug-range:   integer | integer – integer
debug-level:   integer
```

where spaces are for reading ease only. For example,

```
–d12        Set flag 12 to level 1
–d12.3      Set flag 12 to level 3
–d3–17      Set flags 3 through 17 to level 1
–d3–17.4    Set flags 3 through 17 to level 4
```

For a complete list of the available debug flags you will have to look at the code (they are too dynamic to keep this documentation up to date).

3.5. Changing the Values of Options

Options can be overridden using the **–o** or **–O** command line flags. For example,

```
/usr/sbin/sendmail –oT2m
```

sets the **T** (timeout) option to two minutes for this run only; the equivalent line using the long option name is

/usr/sbin/sendmail -OQueueTimeout=2m

Some options have security implications. Sendmail allows you to set these, but relinquishes its setuid root permissions thereafter[10].

3.6. Trying a Different Configuration File

An alternative configuration file can be specified using the **–C** flag; for example,

/usr/sbin/sendmail –Ctest.cf –oQ/tmp/mqueue

uses the configuration file *test.cf* instead of the default */etc/sendmail.cf*. If the **–C** flag has no value it defaults to *sendmail.cf* in the current directory.

Sendmail gives up its setuid root permissions when you use this flag, so it is common to use a publicly writable directory (such as /tmp) as the spool directory (QueueDirectory or Q option) while testing.

3.7. Logging Traffic

Many SMTP implementations do not fully implement the protocol. For example, some personal computer based SMTPs do not understand continuation lines in reply codes. These can be very hard to trace. If you suspect such a problem, you can set traffic logging using the **–X** flag. For example,

/usr/sbin/sendmail –X /tmp/traffic –bd

will log all traffic in the file */tmp/traffic*.

This logs a lot of data very quickly and should **NEVER** be used during normal operations. After starting up such a daemon, force the errant implementation to send a message to your host. All message traffic in and out of *sendmail*, including the incoming SMTP traffic, will be logged in this file.

3.8. Testing Configuration Files

When you build a configuration table, you can do a certain amount of testing using the ‘‘test mode’’ of *sendmail*. For example, you could invoke *sendmail* as:

sendmail –bt –Ctest.cf

which would read the configuration file ‘‘test.cf’’ and enter test mode. In this mode, you enter lines of the form:

rwset address

where *rwset* is the rewriting set you want to use and *address* is an address to apply the set to. Test mode shows you the steps it takes as it proceeds, finally showing you the address it ends up with. You may use a comma separated list of rwsets for sequential application of rules to an input. For example:

3,1,21,4 monet:bollard

first applies ruleset three to the input ‘‘monet:bollard.’’ Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the ‘‘–d21’’ flag to turn on more debugging. For example,

[10]That is, it sets its effective uid to the real uid; thus, if you are executing as root, as from root’s crontab file or during system startup the root permissions will still be honored.

```
sendmail –bt –d21.99
```

turns on an incredible amount of information; a single word address is probably going to print out several pages worth of information.

You should be warned that internally, *sendmail* applies ruleset 3 to all addresses. In test mode you will have to do that manually. For example, older versions allowed you to use

```
0 bruce@broadcast.sony.com
```

This version requires that you use:

```
3,0 bruce@broadcast.sony.com
```

As of version 8.7, some other syntaxes are available in test mode:

- .D x value defines macro *x* to have the indicated *value*. This is useful when debugging rules that use the **$&***x* syntax.
- .C c value adds the indicated *value* to class *c*.
- .S ruleset dumps the contents of the indicated ruleset.
- –d debug-spec is equivalent to the command-line flag.

4. TUNING

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line ‘‘O Timeout.queuereturn=5d’’ sets option ‘‘Timeout.queuereturn’’ to the value ‘‘5d’’ (five days).

Most of these options have appropriate defaults for most sites. However, sites having very high mail loads may find they need to tune them as appropriate for their mail load. In particular, sites experiencing a large number of small messages, many of which are delivered to many recipients, may find that they need to adjust the parameters dealing with queue priorities.

All versions of *sendmail* prior to 8.7 had single character option names. As of 8.7, options have long (multi-character names). Although old short names are still accepted, most new options do not have short equivalents.

This section only describes the options you are most likely to want to tweak; read section 5 for more details.

4.1. Timeouts

All time intervals are set using a scaled syntax. For example, ‘‘10m’’ represents ten minutes, whereas ‘‘2h30m’’ represents two and a half hours. The full set of scales is:

s	seconds
m	minutes
h	hours
d	days
w	weeks

4.1.1. Queue interval

The argument to the **–q** flag specifies how often a sub-daemon will run the queue. This is typically set to between fifteen minutes and one hour. RFC 1123 section 5.3.1.1 recommends that this be at least 30 minutes.

4.1.2. Read timeouts

Timeouts all have option names ‘‘Timeout.*suboption*’’. The recognized *suboption*s, their default values, and the minimum values allowed by RFC 1123 section 5.3.2 are:

connect	The time to wait for an SMTP connection to open (the *connect*(2) system call) [0, unspecified]. If zero, uses the kernel default. In no case can this option extend the timeout longer than the kernel provides, but it can shorten it. This is to get around kernels that provide an absurdly long connection timeout (90 minutes in one case).
initial	The wait for the initial 220 greeting message [5m, 5m].
helo	The wait for a reply from a HELO or EHLO command [5m, unspecified]. This may require a host name lookup, so five minutes is probably a reasonable minimum.
mail†	The wait for a reply from a MAIL command [10m, 5m].
rcpt†	The wait for a reply from a RCPT command [1h, 5m]. This should be long because it could be pointing at a list that takes a long time to expand (see below).
datainit†	The wait for a reply from a DATA command [5m, 2m].
datablock†	The wait for reading a data block (that is, the body of the message). [1h, 3m]. This should be long because it also applies to programs piping input to *sendmail* which have no guarantee of promptness.
datafinal†	The wait for a reply from the dot terminating a message. [1h, 10m]. If this is shorter than the time actually needed for the receiver to deliver the message, duplicates will be generated. This is discussed in RFC 1047.
rset	The wait for a reply from a RSET command [5m, unspecified].
quit	The wait for a reply from a QUIT command [2m, unspecified].
misc	The wait for a reply from miscellaneous (but short) commands such as NOOP (no-operation) and VERB (go into verbose mode). [2m, unspecified].
command†	In server SMTP, the time to wait for another command. [1h, 5m].
ident	The timeout waiting for a reply to an IDENT query [30s[11], unspecified].

For compatibility with old configuration files, if no *suboption* is specified, all the timeouts marked with † are set to the indicated value.

Many of the RFC 1123 minimum values may well be too short. *Sendmail* was designed to the RFC 822 protocols, which did not specify read timeouts; hence, versions of *sendmail* prior to version 8.1 did not guarantee to reply to messages promptly. In particular, a ‘‘RCPT’’ command specifying a mailing list will expand and verify the entire list; a large list on a slow system may easily take more than five minutes[12]. I recommend a one hour timeout — since a communications failure during the RCPT phase is rare, a long timeout is not onerous and may ultimately help reduce network load and duplicated messages.

For example, the lines:

```
O Timeout.command=25m
O Timeout.datablock=3h
```

sets the server SMTP command timeout to 25 minutes and the input data block timeout to three hours.

[11]On some systems the default is zero to turn the protocol off entirely.

[12]This verification includes looking up every address with the name server; this involves network delays, and can in some cases can be considerable.

4.1.3. Message timeouts

After sitting in the queue for a few days, a message will time out. This is to insure that at least the sender is aware of the inability to send a message. The timeout is typically set to five days. It is sometimes considered convenient to also send a warning message if the message is in the queue longer than a few hours (assuming you normally have good connectivity; if your messages normally took several hours to send you wouldn't want to do this because it wouldn't be an unusual event). These timeouts are set using the **Timeout.queuereturn** and **Timeout.queuewarn** options in the configuration file (previously both were set using the **T** option).

Since these options are global, and since you can not know *a priori* how long another host outside your domain will be down, a five day timeout is recommended. This allows a recipient to fix the problem even if it occurs at the beginning of a long weekend. RFC 1123 section 5.3.1.1 says that this parameter should be ''at least 4–5 days''.

The **Timeout.queuewarn** value can be piggybacked on the **T** option by indicating a time after which a warning message should be sent; the two timeouts are separated by a slash. For example, the line

```
OT5d/4h
```

causes email to fail after five days, but a warning message will be sent after four hours. This should be large enough that the message will have been tried several times.

4.2. Forking During Queue Runs

By setting the **ForkEachJob** (**Y**) option, *sendmail* will fork before each individual message while running the queue. This will prevent *sendmail* from consuming large amounts of memory, so it may be useful in memory-poor environments. However, if the **ForkEachJob** option is not set, *sendmail* will keep track of hosts that are down during a queue run, which can improve performance dramatically.

If the **ForkEachJob** option is set, *sendmail* can not use connection caching.

4.3. Queue Priorities

Every message is assigned a priority when it is first instantiated, consisting of the message size (in bytes) offset by the message class (which is determined from the Precedence: header) times the ''work class factor'' and the number of recipients times the ''work recipient factor.'' The priority is used to order the queue. Higher numbers for the priority mean that the message will be processed later when running the queue.

The message size is included so that large messages are penalized relative to small messages. The message class allows users to send ''high priority'' messages by including a ''Precedence:'' field in their message; the value of this field is looked up in the **P** lines of the configuration file. Since the number of recipients affects the amount of load a message presents to the system, this is also included into the priority.

The recipient and class factors can be set in the configuration file using the **RecipientFactor** (**y**) and **ClassFactor** (**z**) options respectively. They default to 30000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

$$pri = msgsize - (class \times \textbf{ClassFactor}) + (nrcpt \times \textbf{RecipientFactor})$$

(Remember, higher values for this parameter actually mean that the job will be treated with lower priority.)

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the ''work time factor,'' set by the **RetryFactor** (**Z**) option. This is added to the priority, so it normally decreases the precedence of the job, on the grounds

that jobs that have failed many times will tend to fail again in the future. The **RetryFactor** option defaults to 90000.

4.4. Load Limiting

Sendmail can be asked to queue (but not deliver) mail if the system load average gets too high using the **QueueLA** (**x**) option. When the load average exceeds the value of the **QueueLA** option, the delivery mode is set to **q** (queue only) if the **QueueFactor** (**q**) option divided by the difference in the current load average and the **QueueLA** option plus one exceeds the priority of the message — that is, the message is queued iff:

$$pri > \frac{\textbf{QueueFactor}}{LA - \textbf{QueueLA} + 1}$$

The **QueueFactor** option defaults to 600000, so each point of load average is worth 600000 priority points (as described above).

For drastic cases, the **RefuseLA** (**X**) option defines a load average at which *sendmail* will refuse to accept network connections. Locally generated mail (including incoming UUCP mail) is still accepted.

4.5. Delivery Mode

There are a number of delivery modes that *sendmail* can operate in, set by the **DeliveryMode** (**d**) configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

- i deliver interactively (synchronously)
- b deliver in background (asynchronously)
- q queue only (don't deliver)
- d defer delvery attempts (don't deliver)

There are tradeoffs. Mode ''i'' gives the sender the quickest feedback, but may slow down some mailers and is hardly ever necessary. Mode ''b'' delivers promptly but can cause large numbers of processes if you have a mailer that takes a long time to deliver a message. Mode ''q'' minimizes the load on your machine, but means that delivery may be delayed for up to the queue interval. Mode ''d'' is identical to mode ''q'' except that it also prevents all the early map lookups from working; it is intended for ''dial on demand'' sites where DNS lookups might cost real money. Some simple error messages (e.g., host unknown during the SMTP protocol) will be delayed using this mode. Mode ''b'' is the usual default.

If you run in mode ''q'' (queue only), ''d'' (defer), or ''b'' (deliver in background) *sendmail* will not expand aliases and follow .forward files upon initial receipt of the mail. This speeds up the response to RCPT commands. Mode ''i'' cannot be used by the SMTP server.

4.6. Log Level

The level of logging can be set for *sendmail*. The default using a standard configuration table is level 9. The levels are as follows:

- 0 No logging.
- 1 Serious system failures and potential security problems.
- 2 Lost communications (network problems) and protocol failures.
- 3 Other serious failures.
- 4 Minor failures.
- 5 Message collection statistics.

6 Creation of error messages, VRFY and EXPN commands.

7 Delivery failures (host or user unknown, etc.).

8 Successful deliveries and alias database rebuilds.

9 Messages being deferred (due to a host being down, etc.).

10 Database expansion (alias, forward, and userdb lookups).

20 Logs attempts to run locked queue files. These are not errors, but can be useful to note if your queue appears to be clogged.

30 Lost locks (only if using lockf instead of flock).

Additionally, values above 64 are reserved for extremely verbose debugging output. No normal site would ever set these.

4.7. File Modes

The modes used for files depend on what functionality you want and the level of security you require.

4.7.1. To suid or not to suid?

Sendmail can safely be made setuid to root. At the point where it is about to *exec*(2) a mailer, it checks to see if the userid is zero; if so, it resets the userid and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the **S** flag to the mailer for mailers that are trusted and must be called as root.) However, this will cause mail processing to be accounted (using *sa*(8)) to root rather than to the user sending the mail.

If you don't make *sendmail* setuid to root, it will still run but you lose a lot of functionality and a lot of privacy, since you'll have to make the queue directory world readable. You could also make *sendmail* setuid to some pseudo-user (e.g., create a user called ''sendmail'' and make *sendmail* setuid to that) which will fix the privacy problems but not the functionality issues. Also, this isn't a guarantee of security: for example, root occasionally sends mail, and the daemon often runs as root.

4.7.2. Should my alias database be writable?

At Berkeley we have the alias database (/etc/aliases*) mode 644. While this is not as flexible as if the database were more 666, it avoids potential security problems with a globally writable database.

The database that *sendmail* actually used is represented by the two files *aliases.dir* and *aliases.pag* (both in /etc) (or *aliases.db* if you are running with the new Berkeley database primitives). The mode on these files should match the mode on /etc/aliases. If *aliases* is writable and the DBM files (*aliases.dir* and *aliases.pag*) are not, users will be unable to reflect their desired changes through to the actual database. However, if *aliases* is read-only and the DBM files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your DBM files are not writable by the world or you do not have auto-rebuild enabled (with the **AutoRebuildAliases** option), then you must be careful to reconstruct the alias database each time you change the text version:

```
newaliases
```

If this step is ignored or forgotten any intended changes will also be ignored or forgotten.

4.8. Connection Caching

When processing the queue, *sendmail* will try to keep the last few open connections open to avoid startup and shutdown costs. This only applies to IPC connections.

When trying to open a connection the cache is first searched. If an open connection is found, it is probed to see if it is still active by sending a NOOP command. It is not an error if this fails; instead, the connection is closed and reopened.

Two parameters control the connection cache. The **ConnectionCacheSize** (**k**) option defines the number of simultaneous open connections that will be permitted. If it is set to zero, connections will be closed as quickly as possible. The default is one. This should be set as appropriate for your system size; it will limit the amount of system resources that *sendmail* will use during queue runs. Never set this higher than 4.

The **ConnectionCacheTimeout** (**K**) option specifies the maximum time that any cached connection will be permitted to idle. When the idle time exceeds this value the connection is closed. This number should be small (under ten minutes) to prevent you from grabbing too many resources from other hosts. The default is five minutes.

4.9. Name Server Access

Control of host address lookups is set by the **hosts** service entry in your service switch file. If you are on a system that has built-in service switch support (e.g., Ultrix, Solaris, or DEC OSF/1) then your system is probably configured properly already. Otherwise, *sendmail* will consult the file **/etc/service.switch**, which should be created. *Sendmail* only uses two entries: **hosts** and **aliases**.

However, some systems (such as SunOS) will do DNS lookups regardless of the setting of the service switch entry. In particular, the system routine *gethostbyname*(3) is used to look up host names, and many vendor versions try some combination of DNS, NIS, and file lookup in /etc/hosts without consulting a service switch. *Sendmail* makes no attempt to work around this problem, and the DNS lookup will be done anyway. If you do not have a nameserver configured at all, such as at a UUCP-only site, *sendmail* will get a ‘‘connection refused’’ message when it tries to connect to the name server. If the **hosts** switch entry has the service ‘‘dns’’ listed somewhere in the list, *sendmail* will interpret this to mean a temporary failure and will queue the mail for later processing; otherwise, it ignores the name server data.

The same technique is used to decide whether to do MX lookups. If you want MX support, you *must* have ‘‘dns’’ listed as a service in the **hosts** switch entry.

The **ResolverOptions** (**I**) option allows you to tweak name server options. The command line takes a series of flags as documented in *resolver*(3) (with the leading ‘‘RES_’’ deleted). Each can be preceded by an optional ‘+’ or ‘–’. For example, the line

```
O ResolverOptions=+AAONLY –DNSRCH
```

turns on the AAONLY (accept authoritative answers only) and turns off the DNSRCH (search the domain path) options. Most resolver libraries default DNSRCH, DEFNAMES, and RECURSE flags on and all others off. You can also include ‘‘HasWildcardMX’’ to specify that there is a wildcard MX record matching your domain; this turns off MX matching when canonifying names, which can lead to inappropriate canonifications.

Version level 1 configurations turn DNSRCH and DEFNAMES off when doing delivery lookups, but leave them on everywhere else. Version 8 of *sendmail* ignores them when doing canonification lookups (that is, when using $[... $]), and always does the search. If you don’t want to do automatic name extension, don’t call $[... $].

The search rules for $[... $] are somewhat different than usual. If the name being looked up has at least one dot, it always tries the unmodified name first. If that fails, it tries the reduced search path, and lastly tries the unmodified name (but only for names without a dot, since names with a dot have already been tried). This allows names such as ‘‘utc.CS’’ to match the site in Czechoslovakia rather than the site in your local Computer Science department. It also prefers A and CNAME records over MX records — that is, if it finds an MX record it makes note of it, but

keeps looking. This way, if you have a wildcard MX record matching your domain, it will not assume that all names match.

To completely turn off all name server access on systems without service switch support (such as SunOS) you will have to recompile with –DNAMED_BIND=0 and remove –lresolv from the list of libraries to be searched when linking.

4.10. Moving the Per-User Forward Files

Some sites mount each user's home directory from a local disk on their workstation, so that local access is fast. However, the result is that .forward file lookups are slow. In some cases, mail can even be delivered on machines inappropriately because of a file server being down. The performance can be especially bad if you run the automounter.

The **ForwardPath** (**J**) option allows you to set a path of forward files. For example, the config file line

O ForwardPath=/var/forward/$u:$z/.forward.$w

would first look for a file with the same name as the user's login in /var/forward; if that is not found (or is inaccessible) the file ''.forward.*machinename*'' in the user's home directory is searched. A truly perverse site could also search by sender by using $r, $s, or $f.

If you create a directory such as /var/forward, it should be mode 1777 (that is, the sticky bit should be set). Users should create the files mode 644.

4.11. Free Space

On systems that have one of the system calls in the *statfs*(2) family (including *statvfs* and *ustat*), you can specify a minimum number of free blocks on the queue filesystem using the **MinFreeBlocks** (**b**) option. If there are fewer than the indicated number of blocks free on the filesystem on which the queue is mounted the SMTP server will reject mail with the 452 error code. This invites the SMTP client to try again later.

Beware of setting this option too high; it can cause rejection of email when that mail would be processed without difficulty.

4.12. Maximum Message Size

To avoid overflowing your system with a large message, the **MaxMessageSize** option can be set to set an absolute limit on the size of any one message. This will be advertised in the ESMTP dialogue and checked during message collection.

4.13. Privacy Flags

The **PrivacyOptions** (**p**) option allows you to set certain ''privacy'' flags. Actually, many of them don't give you any extra privacy, rather just insisting that client SMTP servers use the HELO command before using certain commands or adding extra headers to indicate possible spoof attempts.

The option takes a series of flag names; the final privacy is the inclusive or of those flags. For example:

O PrivacyOptions=needmailhelo, noexpn

insists that the HELO or EHLO command be used before a MAIL command is accepted and disables the EXPN command.

The flags are detailed in section 5.1.6.

4.14. Send to Me Too

Normally, *sendmail* deletes the (envelope) sender from any list expansions. For example, if ''matt'' sends to a list that contains ''matt'' as one of the members he won't get a copy of the message. If the **–m** (me too) command line flag, or if the **MeToo** (**m**) option is set in the configuration file, this behaviour is suppressed. Some sites like to run the SMTP daemon with **–m**.

5. THE WHOLE SCOOP ON THE CONFIGURATION FILE

This section describes the configuration file in detail.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, since this is done every time *sendmail* starts up, rather than easy for a human to read or write. On the ''future project'' list is a configuration-file compiler.

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a sharp symbol ('#') are comments.

5.1. R and S — Rewriting Rules

The core of address parsing are the rewriting rules. These are an ordered production system. *Sendmail* scans through the set of rewriting rules looking for a match on the left hand side (LHS) of the rule. When a rule matches, the address is replaced by the right hand side (RHS) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands are:

S*n*

Sets the current ruleset being collected to *n*. If you begin a ruleset more than once it deletes the old definition.

R*lhs rhs comments*

The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

Macro expansions of the form **$***x* are performed when the configuration file is read. Expansions of the form **$&***x* are performed at run time using a somewhat less general algorithm. This for is intended only for referencing internally defined macros such as **$h** that are changed at runtime.

5.1.1. The left hand side

The left hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

- **$*** Match zero or more tokens
- **$+** Match one or more tokens
- **$–** Match exactly one token
- **$=***x* Match any phrase in class *x*
- **$~***x* Match any word not in class *x*

If any of these match, they are assigned to the symbol **$***n* for replacement on the right hand side, where *n* is the index in the LHS. For example, if the LHS:

$–:$+

is applied to the input:

UCBARPA:eric

the rule will match, and the values passed to the RHS will be:

$1 UCBARPA
$2 eric

Additionally, the LHS can include **$@** to match zero tokens. This is *not* bound to a **$***n* on the RHS, and is normally only used when it stands alone in order to match the null input.

5.1.2. The right hand side

When the left hand side of a rewriting rule matches, the input is deleted and replaced by the right hand side. Tokens are copied directly from the RHS unless they begin with a dollar sign. Metasymbols are:

$*n*	Substitute indefinite token *n* from LHS
$[***name***$]	Canonicalize *name*
$(*map key* **$@***arguments* **$:***default* **$)**	Generalized keyed mapping function
$>*n*	‘‘Call’’ ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*

The **$***n* syntax substitutes the corresponding value from a **$+**, **$–**, **$***, **$=**, or **$~** match on the LHS. It may be used anywhere.

A host name enclosed between **$[** and **$]** is looked up in the host database(s) and replaced by the canonical name[13]. For example, ‘‘$[ftp$]’’ might become ‘‘ftp.CS.Berkeley.EDU’’ and ‘‘$[[128.32.130.2]$]’’ would become ‘‘vangogh.CS.Berkeley.EDU.’’ *Sendmail* recognizes it’s numeric IP address without calling the name server and replaces it with it’s canonical name.

The **$(** ... **$)** syntax is a more general form of lookup; it uses a named map instead of an implicit map. If no lookup is found, the indicated *default* is inserted; if no default is specified and no lookup matches, the value is left unchanged. The *arguments* are passed to the map for possible use.

The **$>***n* syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule. The **$>** syntax can only be used at the beginning of the right hand side; it can be only be preceded by **$@** or **$:**.

The **$#** syntax should *only* be used in ruleset zero or a subroutine of ruleset zero. It causes evaluation of the ruleset to terminate immediately, and signals to *sendmail* that the address has completely resolved. The complete syntax is:

$#*mailer* **$@***host* **$:***user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local the host part may be omitted[14]. The *mailer* must be a single word, but the *host* and *user*

[13]This is actually completely equivalent to $(host *hostname*$). In particular, a **$:** default can be used.

[14]You may want to use it for special ‘‘per user’’ extensions. For example, in the address ‘‘jgm+foo@CMU.EDU’’; the ‘‘+foo’’ part is not part of the user name, and is passed to the local mailer for local use.

may be multi-part. If the *mailer* is the builtin IPC mailer, the *host* may be a colon-separated list of hosts that are searched in order for the first working address (exactly like MX records). The *user* is later rewritten by the mailer-specific envelope rewriting set and assigned to the **$u** macro. As a special case, if the value to **$#** is