SENDMAIL

INSTALLATION AND OPERATION GUIDE

Eric Allman
University of California, Berkeley
Mammoth Project

Version 5.13

For Sendmail Version 5.61

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.

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. The appendixes give a brief but detailed explanation of a number of features not described in the rest of the paper.

The references in this paper are actually found in the companion paper *Sendmail – An Internetwork Mail Router.* This other paper should be read before this manual to gain a basic understanding of how the pieces fit together.

---

*UNIX is a trademark of Bell Laboratories.

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.lib/sendmail* on 4.3BSD.

1.1. Off-The-Shelf Configurations

Configuration files currently in use at Berkeley are in the directory *cf* of the sendmail directory. This directory contains three subdirectories: *cf*, *m4,* and *sitedep*. The directory *cf/m4* contains site-independent *m4*(1) include files that have information common to all configuration files, while *cf/sitedep* contains *m4*(1) include files that have site-specific information in them. These files are used by the master configuration (‘‘.mc’’) in *cf/cf* and produce standard configuration files (with ‘‘.cf’’ suffix) when run through *m4*(1).

Three off the shelf configurations are supplied to handle the basic cases:

(1) Internet sites running the nameserver (or using host tables wherein the fully-qualfied domain name of each host is listed first) can use *cf/tcpproto.cf*. For simple sites, you should be able to use this file without modification. This file is not in *m4* format.

(2) UUCP only sites can use *cf/uucpproto.cf*. This file is not in *m4* format.

(3) A group of machines at a single site connected by an ethernet (or other networking that supports TCP/IP) with (only) one host connected to the outside world via UUCP is represented by two configuration files: *cf/tcpuucpproto.cf* should be installed on the host with outside connections and *cf/tcpproto.cf* should be installed on all other hosts.

Some configuration will be needed in each of the above cases. Just be sure to correctly fill in the ‘‘blanks’’ as shown in the instructions in the configuration file. Then install the file as */usr/lib/sendmail.cf.*

If you are running a larger or more complex site, it is to your advantage to read the ‘‘README’’ file in the *cf* subdirectory. This file explains how to use *m4*(1) to automatically create configuration files for non-standard situations.

1.2. Installation Using the Makefile

A makefile exists in the root of the *sendmail* directory that will do all of these steps for a 4.3BSD system. It may have to be slightly tailored for use on other systems.

Before using this makefile, you should create a symbolic link from *cf* to the directory containing your configuration files. You should also have created your configuration file and left it in the file ‘‘cf/*system*.cf’’ where *system* is the name of your system (i.e., what is returned by *hostname* (1)). If you do not have *hostname* you can use the declaration ‘‘HOST=*system*’’ on the *make* (1) command line. You should also examine the file *md/config.m4* and change the *m4* macros there to reflect any libraries and compilation flags you may need.

The basic installation procedure is to type:

```
make
make install
make installcf
```

in the root directory of the *sendmail* distribution. This will make all binaries and install them in the standard places. The second and third *make* commands must be executed as the superuser (root).

1.3. Installation by Hand

Along with building a configuration file, you will have to install the *sendmail* startup into your UNIX system. If you are doing this installation in conjunction with a regular Berkeley UNIX install, these steps will already be complete. Many of these steps will have to be executed as the superuser (root).

1.3.1. /usr/lib/sendmail

The binary for sendmail is located in /usr/lib. If it becomes necessary to recompile and reinstall the entire system, the following sequence will do it:

```
cd src
make clean
make install
```

1.3.2. /usr/lib/sendmail.cf

The configuration file that you created earlier should be installed in /usr/lib/sendmail.cf:

cp cf/*system*.cf /usr/lib/sendmail.cf

1.3.3. /usr/ucb/newaliases

If you are running delivermail, it is critical that the *newaliases* command be replaced. This can just be a link to *sendmail*:

```
rm –f /usr/ucb/newaliases
ln /usr/lib/sendmail /usr/ucb/newaliases
```

1.3.4. /usr/spool/mqueue

The directory */usr/spool/mqueue* should be created to hold the mail queue. This directory should be mode 755 and owned by root.

1.3.5. /usr/lib/aliases*

The system aliases are held in three files. The file ''/usr/lib/aliases'' is the master copy. A sample is given in ''lib/aliases'' which includes some aliases which *must* be defined:

```
cp lib/aliases /usr/lib/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) routines. These are stored in ''/usr/lib/aliases.dir'' and ''/usr/lib/aliases.pag.'' These can initially be created as empty files, but they will have to be initialized promptly. These should be mode 644:

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

1.3.6. /usr/lib/sendmail.fc

If you intend to install the frozen version of the configuration file (for quick startup) you should create the file /usr/lib/sendmail.fc and initialize it. This step may be safely skipped.

```
cp /dev/null /usr/lib/sendmail.fc
/usr/lib/sendmail –bz
```

1.3.7. /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.

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/lib/sendmail ]; then
        (cd /usr/spool/mqueue; rm −f [lnx]f*)
        /usr/lib/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.

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

1.3.8. /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
```

1.3.9. /usr/lib/sendmail.st

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

```
cp /dev/null /usr/lib/sendmail.st
chmod 666 /usr/lib/sendmail.st
```

This file does not grow. It is printed with the program ‘‘aux/mailstats.’’

1.3.10. /usr/ucb/newaliases

If *sendmail* is invoked as ‘‘newaliases,’’ it will simulate the **−bi** flag (i.e., will rebuild the alias database; see below). This should be a link to /usr/lib/sendmail.

1.3.11. /usr/ucb/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/lib/sendmail.

2. NORMAL OPERATIONS

2.1. Quick Configuration Startup

A fast version of the configuration file may be set up by using the **−bz** flag:

```
/usr/lib/sendmail −bz
```

This creates the file */usr/lib/sendmail.fc* (‘‘frozen configuration’’). This file is an image of *sendmail*’s data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf sendmail.fc* must be rebuilt manually every time *sendmail.cf* is changed.

The frozen configuration file will be ignored if a **−C** flag is specified or if sendmail detects that it is out of date. However, the heuristics are not strong so this should not be trusted.

2.2. The System Log

The system log is supported by the *syslogd* (8) program.

2.2.1. 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 ‘‘sendmail:’’, and a message.

2.2.2. 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 ‘‘useful;’’ log levels above ten are usually for debugging purposes.

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

2.3. The Mail Queue

The mail queue should 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. Format of queue files

All queue files have the form *x* **f***AA99999* where *AA99999* is the *id* for this file and the *x* is a type. The types are:

- d The data file. The message body (excluding the header) is kept in this file.
- l The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous **lf** file can cause a job to apparently disappear (it will not even time out!).
- n This file is created when an id is being created. It is a separate file to insure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.
- q The queue control file. This file contains the information necessary to process the job.
- t A temporary file. These are an image of the **qf** file when it is being rebuilt. It should be renamed to a **qf** file very quickly.
- x A transcript file, existing during the life of a session showing everything that happens during that session.

The **qf** file is structured as a series of lines each beginning with a code letter. The lines are as follows:

- D The name of the data file. There may only be one of these lines.
- H A header definition. There may be any number of these lines. The order is important: they represent the order in the final message. These use the same syntax as header definitions in the configuration file.

R A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.

S The sender address. There may only be one of these lines.

E An error address. If any such lines exist, they represent the addresses that should receive error messages.

T The job creation time. This is used to compute when to time out the job.

P The current message priority. This is used to order the queue. Higher numbers mean lower priorities. The priority changes as the message sits in the queue. The initial priority depends on the message class and the size of the message.

M A message. This line is printed by the *mailq* command, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to ‘‘mckusick@calder’’ and ‘‘wnj’’:

```
DdfA13557
Seric
T404261372
P132
Rmckusick@calder
Rwnj
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.EDU>
Hreceived: by UCBARPA.BERKELEY.EDU (3.227 [10/22/82])
        id A13557; 23-Oct-82 15:49:32-PDT (Sat)
HTo: mckusick@calder, wnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

2.3.3. 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. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without violating the protocol.

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 /usr/spool
mv mqueue omqueue; mkdir mqueue; chmod 755 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/lib/sendmail –oQ/usr/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 /usr/spool/omqueue

2.4. The Alias Database

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

name: name1, name2, ...

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

eric@mit-xx: eric@berkeley.EDU

will not have the desired effect. 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 *dbm*(3) library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag.* This is the form that *sendmail* actually uses to resolve aliases. This technique is used to improve performance.

2.4.1. Rebuilding the alias database

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

newaliases

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

/usr/lib/sendmail –bi

If the ‘‘D’’ option is specified in the configuration, *sendmail* will rebuild the alias database automatically if possible when it is out of date. The conditions under which it will do this are:

(1) The DBM version of the database is mode 666. -or-

(2) *Sendmail* is running setuid to root.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

2.4.2. 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 two 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, 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[1]. *Sendmail* will wait for this entry to appear, at which point it will force a rebuild itself[2].

2.4.3. 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: eric@ucbarpa
```

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.

2.5. 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.

2.6. 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.

2.6.1. Return-Receipt-To:

If this header is sent, a message will be sent to any specified addresses when the final delivery is complete, that is, when successfully delivered to a mailer with the **l** flag (local delivery) set in the mailer descriptor.

2.6.2. Errors-To:

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

2.6.3. Apparently-To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line) then *sendmail* will 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.

At least one recipient line is required under RFC 822.

3. ARGUMENTS

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

---

[1]The ‘‘a’’ option is required in the configuration for this action to occur. This should normally be specified unless you are running *delivermail* in parallel with *sendmail.*

[2]Note: the ‘‘D’’ option must be specified in the configuration file for this operation to occur. If the ‘‘D’’ option is not specified, a warning message is generated and *sendmail* continues.

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 in mode **f** or **a** 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.

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/lib/sendmail −bd −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/lib/sendmail −q −v
```

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. Trying a Different Configuration File

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

```
/usr/lib/sendmail −Ctest.cf
```

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

3.6. Changing the Values of Options

Options can be overridden using the **−o** flag. For example,

```
/usr/lib/sendmail −oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

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 ‘‘OT3d’’ sets option ‘‘T’’ to the value ‘‘3d’’ (three days).

Most of these options default appropriately 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.

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 subdaemon will run the queue. This is typically set to between fifteen minutes and one hour.

4.1.2. Read timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large in certain environments (such as an hour). This will reduce the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

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 three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example,

```
/usr/lib/sendmail –oT1d –q
```

will run the queue and flush anything that is one day old.

4.2. Forking During Queue Runs

By setting the **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 **Y** option is not set, *sendmail* will keep track of hosts that are down during a queue run, which can improve performance dramatically.

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 times the ‘‘work class factor’’ and the number of recipients times the ‘‘work recipient factor.’’ The priority plus the creation time of the message (in seconds since January 1, 1970) are 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 **y** and **z** options respectively. They default to 1000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

pri = size − (class * z) + (nrcpt * y)

(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 **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.

4.4. Load Limiting

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

$$pri > \frac{QF}{LA - x + 1}$$

The **q** option defaults to 10000, so each point of load average is worth 10000 priority points (as described above, that is, bytes + seconds + offsets).

For drastic cases, the **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 ‘‘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)

There are tradeoffs. Mode ‘‘i’’ passes the maximum amount of information to the sender, but is hardly ever necessary. Mode ‘‘q’’ puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval. Mode ‘‘b’’ is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

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 Major problems only.

2 Message collections and failed deliveries.

3 Successful deliveries.

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

5 Normal message queueups.

6 Unusual but benign incidents, e.g., trying to process a locked queue file.

9 Log internal queue id to external message id mappings. This can be useful for tracing a message as it travels between several hosts.

12 Several messages that are basically only of interest when debugging.

16 Verbose information regarding the queue.

4.7. File Modes

There are a number of files that may have a number of modes. The modes 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.

4.7.2. Should my alias database be writable?

At Berkeley we have the alias database (/usr/lib/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 /usr/lib). The mode on these files should match the mode on /usr/lib/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 ‘‘D’’ 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.

5. THE WHOLE SCOOP ON THE CONFIGURATION FILE

This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

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.

An overview of the configuration file is given first, followed by details of the semantics.

5.1. The Syntax

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.

5.1.1. 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.

5.1.2. D − define macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of upper case letters only. Lower case letters and special symbols are used internally.

The syntax for macro definitions is:

D*x val*

where *x* is the name of the macro and *val* is the value it should have. Macros can be interpolated in most places using the escape sequence **$***x*.

5.1.3. C and F − define classes

Classes of words may be defined to match on the left hand side of rewriting rules, where a ‘‘word’’ is a sequence of characters that do not contain characters in the $o macro. For example a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of upper case letters. Lower case letters and special characters are reserved for system use.

The syntax is:

C*c word1 word2...*
F*c file*

The first form defines the class *c* to match any of the named words. It is permissible to split them among multiple lines; for example, the two forms:

CHmonet ucbmonet

and

CHmonet
CHucbmonet

are equivalent. The second form reads the elements of the class *c* from the named *file*.

5.1.4. M − define mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name*, {*field*=*value* }*

where *name* is the name of the mailer (used internally only) and the ‘‘field=name’’ pairs define attributes of the mailer. Fields are:

Path	The pathname of the mailer
Flags	Special flags for this mailer
Sender	A rewriting set for sender addresses
Recipient	A rewriting set for recipient addresses
Argv	An argument vector to pass to this mailer
Eol	The end-of-line string for this mailer
Maxsize	The maximum message length to this mailer

Only the first character of the field name is checked.

5.1.5. H − define header

The format of the header lines that sendmail inserts into the message are defined by the **H** line. The syntax of this line is:

H[**?***mflags***?**]*hname***:** *htemplate*

Continuation lines in this spec are reflected directly into the outgoing message. The *htemplate* is macro expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input it is reflected to the output regardless of these flags.

Some headers have special semantics that will be described below.

5.1.6. O − set option

There are a number of ‘‘random’’ options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*o value*

This sets option *o* to be *value*. Depending on the option, *value* may be a string, an integer, a boolean (with legal values ‘‘t’’, ‘‘T’’, ‘‘f’’, or ‘‘F’’; the default is TRUE), or a time interval.

5.1.7. T − define trusted users

Trusted users are those users who are permitted to override the sender address using the **–f** flag. These typically are ‘‘root,’’ ‘‘uucp,’’ and ‘‘network,’’ but on some users it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T*user1 user2*...

There may be more than one of these lines.

5.1.8. P − precedence definitions

Values for the ‘‘Precedence:’’ field may be defined using the **P** control line. The syntax of this field is:

P*name***=***num*

When the *name* is found in a ‘‘Precedence:’’ field, the message class is set to *num*. Higher numbers mean higher precedence. Numbers less than zero have the special property that error messages will not be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=–100
```

5.2. The Semantics

This section describes the semantics of the configuration file.

5.2.1. Special macros, conditionals

Macros are interpolated using the construct **$***x*, where *x* is the name of the macro to be interpolated. In particular, lower case letters are reserved to have special semantics, used to pass information in or out of sendmail, and some special characters are reserved to provide conditionals, etc.

Conditionals can be specified using the syntax:

```
$?x text1 $| text2 $.
```

This interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The ‘‘else’’ (**$|**) clause may be omitted.

The following macros *must* be defined to transmit information into *sendmail:*

e	The SMTP entry message
j	The ‘‘official’’ domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The set of "operators" in addresses
q	default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants except under terribly unusual circumstances. The **$o** macro consists of a list of characters which will be considered tokens and which will separate tokens when doing parsing. For example, if ‘‘@’’ were in the **$o** macro, then the input ‘‘a@b’’ would be scanned as three tokens: ‘‘a,’’ ‘‘@,’’ and ‘‘b.’’ Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system these definitions are:

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is ‘‘$?x$x $.<$g>’’. These correspond to the following two formats:

```
eric@Berkeley (Eric Allman)
Eric Allman <eric@Berkeley>
```

Some macros are defined by *sendmail* for interpolation into argv’s for mailers or for other contexts. These macros are:

a	The origination date in RFC 822 format
b	The current date in RFC 822 format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	Sendmail's pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of sendmail
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in RFC 822 format; **$a** is the time as extracted from the ‘‘Date:’’ line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no ‘‘Date:’’ line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender as originally determined; when mailing to a specific host the **$g** macro is set to the address of the sender *relative to the recipient.* For example, if I send to ‘‘bollard@matisse’’ from the machine ‘‘ucbarpa’’ the **$f** macro will be ‘‘eric’’ and the **$g** macro will be ‘‘eric@ucbarpa.’’

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as flag to *sendmail.* The second choice is the value of the ‘‘Full-name:’’ line in the header if it exists, and the third choice is the comment field of a ‘‘From:’’ line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

When sending, the **$h**, **$u**, and **$z** macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the **$@** and **$:** part of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (e.g., for the ‘‘Message-Id:’’ field). The **$i** macro is set to the queue id on this host; if put into the timestamp line it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of *sendmail*; this is normally put in timestamps and has been proven extremely useful for debugging. The **$w** macro is set to the name of this host if it can be determined. The **$c** field is set to the ‘‘hop count,’’ i.e., the number of times this message has been processed. This can be determined by the **–h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with sendmail and the sending hostname; these are not supported in the current version.

5.2.2. Special classes

The class **$=w** is set to be the set of all names this host is known by. This can be used to match local hostnames.

5.2.3. 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 token in class *x*
$˜*x*	Match any token 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

5.2.4. 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*
$>*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 using the *gethostent* (3) routines and replaced by the canonical name. For example, ‘‘$[csam$]’’ might become ‘‘lbl-csam.arpa’’ and ‘‘$[[128.32.130.2]$]’’ would become ‘‘vangogh.berkeley.edu.’’

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 should *only* be used in 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. The *mailer* and *host* must be a single word, but the *user* may be multi-part.

A RHS may also be preceded by a **$@** or a **$:** to control evaluation. A **$@** prefix causes the ruleset to return with the remainder of the RHS as the value. A **$:** prefix causes the rule to terminate immediately, but the ruleset to continue; this can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The **$@** and **$:** prefixes may precede a **$>** spec; for example:

```
R$+	$:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the **$:** is necessary to avoid an infinite loop.

Substitution occurs in the order described, that is, parameters from the LHS are substituted, hostnames are canonicalized, ‘‘subroutines’’ are called, and finally **$#**, **$@**, and **$:** are processed.

5.2.5. Semantics of rewriting rule sets

There are five rewriting sets that have specific semantics. These are related as depicted by figure 2.

Ruleset three should turn the address into ‘‘canonical form.’’ This form should have the basic syntax:

local-part@host-domain-spec

If no ‘‘@’’ sign is specified, then the host-domain-spec *may* be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the *sending* mailer). Ruleset three is applied by sendmail before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are going to actually specify recipients. It must resolve to a *{mailer, host, user}* triple. The *mailer* must be defined in the mailer definitions from the configuration file. The *host* is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses respectively. They are applied before any specification in the mailer definition. They must never resolve.

```
boxwid = 0.3i
boxht = 0.3i
movewid = 0.3i
moveht = 0.3i
linewid = 0.3i
lineht = 0.3i

        box invis "addr"; arrow
Box3:   box "3"
A1:     arrow
BoxD:   box "D"; line; L1: Here
C:      [
        C1:arrow; box "1"; arrow; box "S"; line; E1: Here
        move to C1 down 0.5; right
        C2:arrow; box "2"; arrow; box "R"; line; E2: Here
        ] with .w at L1 + (0.5, 0)
        move to C.e right 0.5
L4:     arrow; box "4"; arrow; box invis "msg"
        line from L1 to C.C1
        line from L1 to C.C2
        line from C.E1 to L4
        line from C.E2 to L4
        move to BoxD.n up 0.6; right
Box0:   arrow; box "0"
        arrow; box invis "resolved address" width 1.3
        line from 1/3 of the way between A1 and BoxD.w to Box0
```

Figure 2 – Rewriting set semantics
D – sender domain addition
S – mailer-specific sender rewriting
R – mailer-specific recipient rewriting

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

5.2.6. Mailer flags etc.

There are a number of flags that may be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. These are detailed in Appendix C. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers.

5.2.7. The ‘‘error’’ mailer

The mailer with the special name ‘‘error’’ can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

$#error$:Host unknown in this domain

on the RHS of a rule will cause the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

5.3. Building a Configuration File From Scratch

Building a configuration table from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what it is that you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain what the real purpose of a configuration table is and to give you some ideas for what your philosophy might be.

5.3.1. What you are trying to do

The configuration table has three major purposes. The first and simplest is to set up the environment for *sendmail*. This involves setting the options, defining a few critical macros, etc. Since these are described in other places, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer. *Sendmail* does this in three subphases. Rulesets one and two are applied to all sender and recipient addresses respectively. After this, you may specify per-mailer rulesets for both sender and recipient addresses; this allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. The mailer descriptor describes the interface requirements of the mailer.

5.3.2. Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. I will present a few possible philosophies here.

One general point applies to all of these philosophies: it is almost always a mistake to try to do full name resolution. For example, if you are trying to get names of the form ‘‘user@host’’ to the Arpanet, it does not pay to route them to ‘‘xyzvax!decvax!ucbvax!c70:user@host’’ since you then depend on several links not under your control. The best approach to this problem is to simply forward to ‘‘xyzvax!user@host’’ and let xyzvax worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

5.3.2.1. Large site, many hosts − minimum information

Berkeley is an example of a large site, i.e., more than two or three hosts and multiple mail connections. We have decided that the only reasonable philosophy in our environment is to designate one host as the guru for our site. It must be able to resolve any piece of mail it receives. The other sites should have the minimum amount of information they can get away with. In addition, any information they do have should be hints rather than solid information.

For example, a typical site on our local ether network is ‘‘monet.’’ When monet receives mail for delivery, it checks whether it knows that the destination host is directly reachable; if so, mail is sent to that host. If it receives mail for any unknown host, it just passes it directly to ‘‘ucbvax,’’ our master host. Ucbvax may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new mail connection is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated if convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, some UUCP connections are currently on ‘‘ucbarpa.’’ However, monet *does not* know about this; the information is hidden totally between ucbvax and ucbarpa. Mail going from monet to a UUCP host is transferred via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to UUCP. Although this involves some extra hops, we feel this is an acceptable tradeoff.

An interesting point is that it would be possible to update monet to send appropriate UUCP mail directly to ucbarpa if the load got too high; if monet failed to note a host as connected to ucbarpa it would go via ucbvax as before, and if monet incorrectly sent a message to ucbarpa it would still be sent by ucbarpa to ucbvax as before. The only problem that can occur is loops, for example, if ucbarpa thought that ucbvax had the UUCP connection and vice versa. For this reason, updates should *always* happen to the master host first.

This philosophy results as much from the need to have a single source for the configuration files (typically built using *m4*(1) or some similar tool) as any logical need. Maintaining more than three separate tables by hand is essentially an impossible job.

5.3.2.2. Small site − complete information

A small site (two or three hosts and few external connections) may find it more reasonable to have complete information at each host. This would require that each host know exactly where each network connection is, possibly including the names of each host on that network. As long as the site remains small and the the configuration remains relatively static, the update problem will probably not be too great.

5.3.2.3. Single host

This is in some sense the trivial case. The only major issue is trying to insure that you don’t have to know too much about your environment. For example, if you have a UUCP connection you might find it useful to know about the names of hosts connected directly to you, but this is really not necessary since this may be determined from the syntax.

5.3.3. Relevant issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822. Copies of these RFC’s are included on the *sendmail* tape as *doc/rfc819.lpr* and *doc/rfc822.lpr*.

RFC822 describes the format of the mail message itself. *Sendmail* follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

    < > ( ) " \

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially each host is given a name which is a right-to-left dot qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Berkeley one legal host might be ‘‘a.CC.Berkeley.EDU’’; reading from right to left, ‘‘EDU’’ is a top level domain comprising educational institutions, ‘‘Berkeley’’ is a logical domain name, ‘‘CC’’ represents the Computer Center, (in this case a strictly logical entity), and ‘‘a’’ is a host in the Computer Center.

Beware when reading RFC819 that there are a number of errors in it.

5.3.4. How to proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide if any of them are close enough to steal major parts of. Even under the worst of conditions, there is a fair amount of boiler plate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since *sendmail* likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Berkeley configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

5.3.5. Testing the rewriting rules − the −bt flag

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; ruleset three is always applied first. For example:

    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,

    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.

5.3.6. Building mailer descriptions

To add an outgoing mailer to your mail system, you will have to define the characteristics of the mailer.

Each mailer must have an internal name. This can be arbitrary, except that the names ‘‘local’’ and ‘‘prog’’ must be defined.

The pathname of the mailer must be given in the P field. If this mailer should be accessed via an IPC connection, use the string ‘‘[IPC]’’ instead.

The F field defines the mailer flags. You should specify an ‘‘f’’ or ‘‘r’’ flag to pass the name of the sender as a **−f** or **−r** flag respectively. These flags are only passed if they were passed to *sendmail*, so that mailers that give errors under some circumstances can be placated. If the mailer is not picky you can just specify ‘‘−f $g’’ in the argv template. If the mailer must be called as **root** the ‘‘S’’ flag should be given; this will not reset the userid before calling the mailer[3]. If this mailer is local (i.e., will perform final delivery rather than another network hop) the ‘‘l’’ flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the ‘‘s’’ flag is specified; if this is not given they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction the ‘‘m’’ flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The ‘‘e’’ flag will mark the mailer as being ‘‘expensive,’’ which will cause *sendmail* to defer connection until a queue run[4].

An unusual case is the ‘‘C’’ flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain spec of the sender (i.e., the ‘‘@host.domain’’ part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

```
From: eric@ucbarpa
To: wnj@monet, mckusick
```

will be modified to:

```
From: eric@ucbarpa
To: wnj@monet, mckusick@ucbarpa
```

if and only if the ‘‘C’’ flag is defined in the mailer corresponding to ‘‘eric@ucbarpa.’’

Other flags are described in Appendix C.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses respectively. These are applied after the sending domain is appended and the general rewriting sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. These sets can also be used to do special purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

Finally, an argv template is given as the E field. It may have embedded spaces. If there is no argv with a **$u** macro in it, *sendmail* will speak SMTP to the mailer. If the pathname for this mailer is ‘‘[IPC],’’ the argv should be

IPC $h [*port*]

---

[3]*Sendmail* must be running setuid to root for this to work.

[4]The ‘‘c’’ configuration option must be given for this to be effective.

where *port* is the optional port number to connect to.

For example, the specifications:

```
Mlocal, P=/bin/mail, F=rlsm  S=10, R=20, A=mail −d $u
Mether, P=[IPC],     F=meC, S=11, R=21, A=IPC $h, M=100000
```

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called ‘‘local,’’ is located in the file ‘‘/bin/mail,’’ takes a picky **−r** flag, does local delivery, quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the message and ruleset twenty should be applied to recipient addresses; the argv to send to a message will be the word ‘‘mail,’’ the word ‘‘−d,’’ and words containing the name of the receiving user. If a **−r** flag is inserted it will be between the words ‘‘mail’’ and ‘‘−d.’’ The second mailer is called ‘‘ether,’’ it should be connected to via an IPC connection, it can handle multiple users at once, connections should be deferred, and any domain from the sender address should be appended to any receiver name without a domain; sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000 byte limit on messages passed through this mailer.

APPENDIX A

COMMAND LINE FLAGS

Arguments must be presented with flags before addresses. The flags are:

−f *addr*	The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a ''trusted user'' or if *addr* contains an exclamation point (because of certain restrictions in UUCP).
−r *addr*	An obsolete form of **−f**.
−h *cnt*	Sets the ''hop count'' to *cnt*. This represents the number of times this message has been processed by *sendmail* (to the extent that it is supported by the underlying networks). *Cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) *sendmail* throws away the message with an error.
−F*name*	Sets the full name of this user to *name*.
−n	Don't do aliasing or forwarding.
−t	Read the header for ''To:'', ''Cc:'', and ''Bcc:'' lines, and send to everyone listed in those lists. The ''Bcc:'' line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.
−b*x*	Set operation mode to *x*. Operation modes are:

m	Deliver mail (default)
a	Run in arpanet mode (see below)
s	Speak SMTP on input side
d	Run as a daemon
t	Run in test mode
v	Just verify addresses, don't collect or deliver
i	Initialize the alias database
p	Print the mail queue
z	Freeze the configuration file

The special processing for the ARPANET includes reading the ''From:'' line from the header to find the sender, printing ARPANET style messages (preceded by three digit reply codes for compatibility with the FTP protocol [Neigus73, Postel74, Postel77]), and ending lines of error messages with <CRLF>.

−q*time*	Try to process the queued up mail. If the time is given, a sendmail will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.
−C*file*	Use a different configuration file. *Sendmail* runs as the invoking user (rather than root) when this flag is specified.
−d*level*	Set debugging level.
−o*x value*	Set option *x* to the specified *value*. These options are described in Appendix B.

There are a number of options that may be specified as primitive flags (provided for compatibility with *delivermail*). These are the e, i, m, and v options. Also, the f option may be specified as the **−s** flag.

APPENDIX B

CONFIGURATION OPTIONS

The following options may be set using the –**o** flag on the command line or the **O** line in the configuration file. Many of them cannot be specified unless the invoking user is trusted.

A*file*	Use the named *file* as the alias file. If no file is specified, use *aliases* in the current directory.
a*N*	If set, wait up to *N* minutes for an ‘‘@:@’’ entry to exist in the alias database before starting up. If it does not appear in *N* minutes, rebuild the database (if the **D** option is also set) or issue a warning.
B*c*	Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character.
c	If an outgoing mailer is marked as being expensive, don’t connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.
d*x*	Deliver in mode *x*. Legal modes are: i Deliver interactively (synchronously) b Deliver in background (asynchronously) q Just queue the message (deliver during queue run)
D	If set, rebuild the alias database if necessary and possible. If this option is not set, *sendmail* will never rebuild the alias database unless explicitly requested using –**bi**.
e*x*	Dispose of errors using mode *x*. The values for *x* are: p Print error messages (default) q No messages, just give exit status m Mail back errors w Write back errors (mail if user not logged in) e Mail back errors and give zero exit stat always
f	Save Unix-style ‘‘From’’ lines at the front of headers. Normally they are assumed redundant and discarded.
g*n*	Set the default group id for mailers to run in to *n*.
H*file*	Specify the help file for SMTP.
I	Insist that the BIND name server be running to resolve host names. If this is not set and the name server is not running, the */etc/hosts* file will be considered complete. In general, you do want to set this option if your */etc/hosts* file does not include all hosts known to you or if you are using the MX (mail forwarding) feature of the BIND name server. The name server will still be consulted even if this option is not set, but *sendmail* will feel free to resort to reading */etc/hosts* if the name server is not available. Thus, you should *never* set this option if you do not run the name server.
i	Ignore dots in incoming messages.
L*n*	Set the default log level to *n*.
M*x value*	Set the macro *x* to *value*. This is intended only for use from the command line.
m	Send to me too, even if I am in an alias expansion.

N*netname* The name of the home network; ‘‘ARPA’’ by default. The the argument of an SMTP ‘‘HELO’’ command is checked against ‘‘hostname.netname’’ where *hostname* is requested from the kernel for the current connection. If they do not match, ‘‘Received:’’ lines are augmented by the name that is determined in this manner so that messages can be traced accurately.

o Assume that the headers may be in old format, i.e., spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.

Q*dir* Use the named *dir* as the queue directory.

q*factor* Use *factor* as the multiplier in the map function to decide when to just queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (**x** flag) to determine the maximum message priority that will be sent. Defaults to 10000.

r*time* Timeout reads after *time* interval.

S*file* Log statistics in the named *file*.

s Be super-safe when running things, i.e., always instantiate the queue file, even if you are going to attempt immediate delivery. *Sendmail* always instantiates the queue file before returning control the the client under any circumstances.

T*time* Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.

t*S,D* Set the local time zone name to *S* for standard time and *D* for daylight time; this is only used under version six.

u*n* Set the default userid for mailers to *n*. Mailers without the *S* flag in the mailer definition will run as this user.

v Run in verbose mode.

x*LA* When the system load average exceeds *LA*, just queue messages (i.e., don’t try to send them).

X*LA* When the system load average exceeds *LA*, refuse incoming SMTP connections.

y*fact* The indicated *fact*or is added to the priority (thus *lowering* the priority of the job) for each recipient, i.e., this value penalizes jobs with large numbers of recipients.

Y If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, since the default tends to consume considerable amounts of memory while the queue is being processed.

z*fact* The indicated *fact*or is multiplied by the message class (determined by the Precedence: field in the user header and the **P** lines in the configuration file) and subtracted from the priority. Thus, messages with a higher Priority: will be favored.

Z*fact* The *fact*or is added to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments this should be positive, since hosts that are down are all too often down for a long time.

APPENDIX C

MAILER FLAGS

The following flags may be set in the mailer description.

f The mailer wants a **–f** *from* flag, but only if this is a network forward operation (i.e., the mailer will give an error if the executing user does not have special permissions).

r Same as **f**, but sends a **–r** flag.

S Don't reset the userid before calling the mailer. This would be used in a secure environment where *sendmail* ran as root. This could be used to avoid forged addresses. This flag is suppressed if given from an ‘‘unsafe’’ environment (e.g, a user's mail.cf file).

n Do not insert a UNIX-style ‘‘From’’ line on the front of the message.

l This mailer is local (i.e., final delivery will be performed).

s Strip quote characters off of the address before calling the mailer.

m This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users.

F This mailer wants a ‘‘From:’’ header line.

D This mailer wants a ‘‘Date:’’ header line.

M This mailer wants a ‘‘Message-Id:’’ header line.

x This mailer wants a ‘‘Full-Name:’’ header line.

P This mailer wants a ‘‘Return-Path:’’ line.

u Upper case should be preserved in user names for this mailer.

h Upper case should be preserved in host names for this mailer.

A This is an Arpanet-compatible mailer, and all appropriate modes should be set.

U This mailer wants Unix-style ‘‘From’’ lines with the ugly UUCP-style ‘‘remote from <host>’’ on the end.

e This mailer is expensive to connect to, so try to avoid connecting normally; any necessary connection will occur during a queue run.

X This mailer want to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This insures that lines in the message containing a dot will not terminate the message prematurely.

L Limit the line lengths as specified in RFC821.

P Use the return-path in the SMTP ‘‘MAIL FROM:’’ command rather than just the return address; although this is required in RFC821, many hosts do not process return paths properly.

I This mailer will be speaking SMTP to another *sendmail* – as such it can use special protocol features. This option is not required (i.e., if this option is omitted the transmission will still operate successfully, although perhaps not as efficiently as possible).

C If mail is *received* from a mailer with this flag set, any addresses in the header that do not have an at sign (‘‘@’’) after being rewritten by ruleset three will have the ‘‘@domain’’ clause from the sender tacked on. This allows mail with headers of the form:

    From: usera@hosta
    To: userb@hostb, userc

to be rewritten as:

From: usera@hosta
To: userb@hostb, userc@hosta

automatically.

E Escape lines beginning with ‘‘From’’ in the message with a ‘>’ sign.

APPENDIX D

OTHER CONFIGURATION

There are some configuration changes that can be made by recompiling *sendmail*. These are located in two places:

src/conf.h — Configuration parameters that may be tweaked by the installer are included in conf.h.

src/conf.c — Some special routines and a few variables may be defined in conf.c. For the most part these are selected from the settings in conf.h.

Parameters in src/conf.h

Parameters and compilation options are defined in conf.h. Most of these need not normally be tweaked; common parameters are all in sendmail.cf. However, the sizes of certain primitive vectors, etc., are included in this file. The numbers following the parameters are their default value.

MAXLINE [1024] — The maximum line length of any input line. If message lines exceed this length they will still be processed correctly; however, header lines, configuration file lines, alias lines, etc., must fit within this limit.

MAXNAME [256] — The maximum length of any name, such as a host or a user name.

MAXFIELD [2500] — The maximum total length of any header field, including continuation lines.

MAXPV [40] — The maximum number of parameters to any mailer. This limits the number of recipients that may be passed in one transaction.

MAXHOP [17] — When a message has been processed more than this number of times, sendmail rejects the message on the assumption that there has been an aliasing loop. This can be determined from the **−h** flag or by counting the number of trace fields (i.e, ‘‘Received:’’ lines) in the message header.

MAXATOM [100] — The maximum number of atoms (tokens) in a single address. For example, the address ‘‘eric@Berkeley’’ is three atoms.

MAXMAILERS [25] — The maximum number of mailers that may be defined in the configuration file.

MAXRWSETS [30] — The maximum number of rewriting sets that may be defined.

MAXPRIORITIES [25] — The maximum number of values for the ‘‘Precedence:’’ field that may be defined (using the **P** line in sendmail.cf).

MAXTRUST [30] — The maximum number of trusted users that may be defined (using the **T** line in sendmail.cf).

MAXUSERENVIRON [40] — The maximum number of items in the user environment that will be passed to subordinate mailers.

QUEUESIZE [600] — The maximum number of entries that will be processed in a single queue run.

A number of other compilation options exist. These specify whether or not specific code should be compiled in.

DBM — If set, the ‘‘DBM’’ package in UNIX is used (see *dbm(3X)* in [UNIX80]). If not set, a much less efficient algorithm for processing aliases is used.

NDBM — If set, the new version of the DBM library that allows multiple databases will be used. ‘‘DBM’’ must also be set.

DEBUG If set, debugging information is compiled in. To actually get the debugging output, the **−d** flag must be used.

LOG If set, the *syslog* routine in use at some sites is used. This makes an informational log record for each message processed, and makes a higher priority log record for internal system errors.

QUEUE This flag should be set to compile in the queueing code. If this is not set, mailers must accept the mail immediately or it will be returned to the sender.

SMTP If set, the code to handle user and server SMTP will be compiled in. This is only necessary if your machine has some mailer that speaks SMTP.

DAEMON If set, code to run a daemon is compiled in. This code is for 4.2 or 4.3BSD.

UGLYUUCP If you have a UUCP host adjacent to you which is not running a reasonable version of *rmail*, you will have to set this flag to include the ‘‘remote from sysname’’ info on the from line. Otherwise, UUCP gets confused about where the mail came from.

NOTUNIX If you are using a non-UNIX mail format, you can set this flag to turn off special processing of UNIX-style ‘‘From ’’ lines.

NAMED_BIND Compile in code to use the Berkeley Internet Name Domain (BIND) server to resolve TCP/IP host names.

SETPROCTITLE If defined, *sendmail* will change its *argv* array to indicate its current status. This can be used in conjunction with the *ps* command to find out just what it’s up to.

NO_WILDCARD_MX
Should be set if there are no wildcard MX nameserver records in the local domain. If set, this will enable the use of ANY query types, resulting in better performance. Unfortunately, wildcard MX records in the local domain will mess this up, hence the need for this compilation option.

Configuration in src/conf.c

Not all header semantics are defined in the configuration file. Header lines that should only be included by certain mailers (as well as other more obscure semantics) must be specified in the *HdrInfo* table in *conf.c*. This table contains the header name (which should be in all lower case) and a set of header control flags (described below), The flags are:

H_ACHECK Normally when the check is made to see if a header line is compatible with a mailer, *sendmail* will not delete an existing line. If this flag is set, *sendmail* will delete even existing header lines. That is, if this bit is set and the mailer does not have flag bits set that intersect with the required mailer flags in the header definition in sendmail.cf, the header line is *always* deleted.

H_EOH If this header field is set, treat it like a blank line, i.e., it will signal the end of the header and the beginning of the message text.

H_FORCE Add this header entry even if one existed in the message before. If a header entry does not have this bit set, *sendmail* will not add another header line if a header line of this name already existed. This would normally be used to stamp the message by everyone who handled it.

H_TRACE If set, this is a timestamp (trace) field. If the number of trace fields in a message exceeds a preset amount the message is returned on the assumption that it has an aliasing loop.

H_RCPT If set, this field contains recipient addresses. This is used by the **−t** flag to determine who to send to when it is collecting recipients from the message.

H_FROM This flag indicates that this field specifies a sender. The order of these fields in the *HdrInfo* table specifies *sendmail’s* preference for which field to return error messages to.

Let’s look at a sample *HdrInfo* specification:

```
struct hdrinfo          HdrInfo[] =
{
        /* originator fields, most to least significant  */
    "resent-sender",    H_FROM,
    "resent-from",      H_FROM,
    "sender",           H_FROM,
    "from",             H_FROM,
    "full-name",        H_ACHECK,
        /* destination fields */
    "to",               H_RCPT,
    "resent-to",        H_RCPT,
    "cc",               H_RCPT,
        /* message identification and control */
    "message",          H_EOH,
    "text",             H_EOH,
        /* trace fields */
    "received",         H_TRACE|H_FORCE,

    NULL,               0,
};
```

This structure indicates that the ‘‘To:’’, ‘‘Resent-To:’’, and ‘‘Cc:’’ fields all specify recipient addresses. Any ‘‘Full-Name:’’ field will be deleted unless the required mailer flag (indicated in the configuration file) is specified. The ‘‘Message:’’ and ‘‘Text:’’ fields will terminate the header; these are specified in new protocols [NBS80] or used by random dissenters around the network world. The ‘‘Received:’’ field will always be added, and can be used to trace messages.

There are a number of important points here. First, header fields are not added automatically just because they are in the *HdrInfo* structure; they must be specified in the configuration file in order to be added to the message. Any header fields mentioned in the configuration file but not mentioned in the *HdrInfo* structure have default processing performed; that is, they are added unless they were in the message already. Second, the *HdrInfo* structure only specifies cliched processing; certain headers are processed specially by ad hoc code regardless of the status specified in *HdrInfo*. For example, the ‘‘Sender:’’ and ‘‘From:’’ fields are always scanned on ARPANET mail to determine the sender; this is used to perform the ‘‘return to sender’’ function. The ‘‘From:’’ and ‘‘Full-Name:’’ fields are used to determine the full name of the sender if possible; this is stored in the macro **$x** and used in a number of ways.

The file *conf.c* also contains the specification of ARPANET reply codes. There are four classifications these fall into:

```
char  Arpa_Info[] =      "050";  /* arbitrary info */
char  Arpa_TSyserr[] =  "455";  /* some (transient) system error */
char  Arpa_PSyserr[] =  "554";  /* some (permanent) system error */
char  Arpa_Usrerr[] =    "554";  /* some (fatal) user error */
```

The class *Arpa_Info* is for any information that is not required by the protocol, such as forwarding information. *Arpa_TSyserr* and *Arpa_PSyserr* is printed by the *syserr* routine. TSyserr is printed out for transient errors, that is, errors that are likely to go away without explicit action on the part of a systems administrator. PSyserr is printed for permanent errors. The distinction is made based on the value of *errno*. Finally, *Arpa_Usrerr* is the result of a user error and is generated by the *usrerr* routine; these are generated when the user has specified something wrong, and hence the error is permanent, i.e., it will not work simply by resubmitting the request.

If it is necessary to restrict mail through a relay, the *checkcompat* routine can be modified. This routine is called for every recipient address. It can return **TRUE** to indicate that the address is acceptable and mail processing will continue, or it can return **FALSE** to reject the recipient. If it returns false, it is up to *checkcompat* to print an error message (using *usrerr*) saying why the message is rejected. For example, *checkcompat* could read:

```
bool
checkcompat(to)
	register ADDRESS *to;
{
	if (MsgSize > 50000 && to->q_mailer != LocalMailer)
	{
		usrerr("Message too large for non-local delivery");
		NoReturn = TRUE;
		return (FALSE);
	}
	return (TRUE);
}
```

This would reject messages greater than 50000 bytes unless they were local. The *NoReturn* flag can be sent to suppress the return of the actual body of the message in the error return. The actual use of this routine is highly dependent on the implementation, and use should be limited.

Configuration in src/daemon.c

The file *src/daemon.c* contains a number of routines that are dependent on the local networking environment. The version supplied is specific to 4.3 BSD.

The routine *maphostname* is called to convert strings within **$[** ... **$]** symbols. It can be modified if you wish to provide a more sophisticated service, e.g., mapping UUCP host names to full paths.

APPENDIX E

SUMMARY OF SUPPORT FILES

This is a summary of the support files that *sendmail* creates or generates.

/usr/lib/sendmail — The binary of *sendmail*.

/usr/bin/newaliases
: A link to /usr/lib/sendmail; causes the alias database to be rebuilt. Running this program is completely equivalent to giving *sendmail* the **−bi** flag.

/usr/bin/mailq — Prints a listing of the mail queue. This program is equivalent to using the **−bp** flag to *sendmail*.

/usr/lib/sendmail.cf
: The configuration file, in textual form.

/usr/lib/sendmail.fc
: The configuration file represented as a memory image.

/usr/lib/sendmail.hf
: The SMTP help file.

/usr/lib/sendmail.st
: A statistics file; need not be present.

/usr/lib/aliases — The textual version of the alias file.

/usr/lib/aliases.{pag,dir}
: The alias file in *dbm* (3) format.

/usr/spool/mqueue
: The directory in which the mail queue and temporary files reside.

/usr/spool/mqueue/qf*
: Control (queue) files for messages.

/usr/spool/mqueue/df*
: Data files.

/usr/spool/mqueue/lf*
: Lock files

/usr/spool/mqueue/tf*
: Temporary versions of the qf files, used during queue file rebuild.

/usr/spool/mqueue/nf*
: A file used when creating a unique id.

/usr/spool/mqueue/xf*
: A transcript of the current session.

TABLE OF CONTENTS