Appendix B

IRIX sendmail Reference

This appendix provides reference material on *sendmail*. It is divided into the following sections:

- "sendmail Command-Line Flags" on page 221
- "Tuning" on page 224
- "The Configuration File" on page 229
- "Flags, Options, and Files" on page 247

sendmail Command-Line Flags

You can include one or more flags on the command line to tailor a *sendmail* session. This section describes some of the more frequently used flags. For a complete description of command-line flags, see "Flags, Options, and Files" on page 247.

Changing the Values of Configuration Options

The **-o** flag overrides an option in the configuration file. The override is for the current session only. In the following example, the **T** (timeout) option becomes two minutes for this session only:

```
/usr/lib/sendmail -oT2m
```

For a complete discussion of configuration options, see "Flags, Options, and Files" on page 247.

Delivery Mode

One configuration option frequently overridden on the command line is the **d** option, which specifies the *sendmail* delivery mode. The delivery mode determines how quickly mail is delivered:

i	deliver interactively (synchronously)
b	deliver in background (asynchronously)
q	queue only (don't deliver)

There are trade-offs. Mode **i** passes the maximum amount of information to the sender, but is rarely necessary.

Mode **q** puts the minimum load on your system, but if you use it, delivery may be delayed for up to the queue interval.

Mode **b** is probably a good compromise. However, in this mode, *sendmail* may initiate a large number of processes if you have a mailer that takes a long time to deliver a message.

Queue Mode

The **-q** flag causes *sendmail* to process the mail queue at regular intervals. The syntax is as follows, where *time* defines the interval between instances of queue processing:

–q [*time*]

Time is expressed in number of minutes: 15m sets the interval to 15 minutes. If *time* is omitted, *sendmail* processes the queue once and returns. The **-q** flag is often used in conjunction with daemon mode, described in the next subsection.

See "Timeouts and Intervals" on page 225 for a discussion of time-interval specifications and formats.

Daemon Mode

To process incoming mail over sockets, a daemon must be running. The **-bd** flag causes *sendmail* to run in daemon mode. The **-bd** and **-q** flags can be combined in one call, as in the following example:

```
/usr/lib/sendmail -bd -q30m
```

This command causes *sendmail* to run in daemon mode and to fork a subdaemon for queue processing every half hour.

The script for starting *sendmail* that is provided with IRIX includes the following command line:

```
/usr/lib/sendmail -bd -q15m
```

Verify Mode

Using the **-bv** flag directs *sendmail* to validate addresses, aliases, and mailing lists. In this mode, *sendmail* performs verification only. It does not try to collect or deliver a message. *sendmail* expands all aliases, suppresses duplicates, and displays the expanded list of names. For each name, *sendmail* indicates if it knows how to deliver a message to that destination.

Test Mode

The **-bt** flag places *sendmail* in test mode so that it describes how the current configuration rewrites addresses. Test mode is extremely useful for debugging modifications to the */usr/lib/sendmail.cf* configuration file. For more information, see "Test Mode" on page 242.

Debugging Flags

Several debugging flags are built into *sendmail.* Each flag includes a number and a level. The number identifies the debugging flag. The level, which defaults to 1, dictates how much information prints. A low level causes minimal information to print; a high level causes more comprehensive information to print. In general, levels greater than 9 cause so much information to print that it is of limited value. Debugging flags use the following syntax:

-d *debug-list*

A debug list includes the flag number and the flag level, as shown in the following examples.

- Set flag 13 to level 1.

-d13

- Set flag 13 to level 3.

  -d13.3

- Set flags 5 though 18 to level 1.

  -d5-18

- Set flags 5 through 18 to level 4.

  -d5-18.4

Many debugging flags are of little use to the average *sendmail* user. Some are occasionally useful for helping to track down obscure problems. "Flags, Options, and Files" on page 247 includes a complete list of debugging flags.

Using a Different Configuration File

The *-C* flag directs *sendmail* to use an alternate configuration file. For example, the following line directs *sendmail* to use the *test.cf* file instead of the default */usr/lib/sendmail.cf* file:

```
/usr/lib/sendmail –Ctest.cf
```

If the *-C* flag appears without a filename, *sendmail* uses the file *sendmail.cf* in the current directory. Thus, the *-C* flag directs *sendmail* to ignore any */usr/lib/sendmail.fc* ("frozen") file that may be present.

Tuning

A number of configuration parameters are available for fine-tuning *sendmail* to the requirements of a specific site. Options in the configuration file set these parameters. For example, the string "T3d" sets the **T** (timeout) option to "3d" (three days).

Most options have default values that are appropriate for many sites. However, sites having very high mail loads may need to tune these parameters to fit the mail load. In particular, sites with a large volume of small messages that are delivered to multiple recipients may need to adjust the parameters for queue priorities.

The rest of this section describes the configuration parameters for the following tuning areas:

- timeouts and intervals
- forking during queue runs
- queue priorities
- load limiting
- delivery mode
- log level

Timeouts and Intervals

Time intervals use the following abbreviations:

s	seconds
m	minutes
h	hours
d	days
w	weeks

For example, “10m” represents 10 minutes, and “2h30m” represents two hours and 30 minutes.

Queue Interval

The argument to the **-q** flag specifies how often to process the mail queue. When the *sendmail* daemon is started with the */etc/init.d/mail* script, the queue interval is set to 15 minutes.

If *sendmail* runs in delivery mode **b**, messages are written to the queue only when they cannot be delivered (for example, when a recipient host is down). Therefore, the need to process the queue is limited and the queue interval value may be set quite high. The value is relevant only when a host that was down comes back up.

If *sendmail* runs in delivery mode **q**, the queue interval should be set to a low value, as it defines the longest time that a message sits in the local queue before being processed.

Read Timeouts

sendmail can time out when reading the standard input or when reading from a remote SMTP server. Technically, a timeout is not acceptable within the published protocols. However, setting the read timeout option to a high value (such as an hour) reduces the chance that a large number of idle daemons will pile up on a system. The read timeout option is **r**.

Message Timeouts

sendmail causes a queued message to time out after a specified time period. This feature ensures that the sender knows the message cannot be delivered. This default message timeout value is one week (seven days). The value is set with the **T** option.

The queue records the time of submission, rather than the time remaining until timeout. This approach enables *sendmail* to flush messages that have been hanging for a short period by running the queue with a short message timeout. The following example illustrates how to process the queue and flush any message that is one day old:

```
/usr/lib/sendmail –oT1d –q
```

Forking During Queue Runs

Setting the **Y** option causes *sendmail* to fork before each individual message when processing the queue. This technique prevents *sendmail* from consuming large amounts of memory, and may be useful in memory-poor environments. However, if the **Y** option is not set, *sendmail* keeps track of hosts that are down during a queue run, which can improve performance dramatically.

Queue Priorities

sendmail assigns a priority to every message when it is first instantiated. *sendmail* uses the priority and the message creation time (in seconds since January 1, 1970) to order the queue. The message with the lowest priority number is processed first. The algorithm that derives a message's priority uses the following information:

message size (in bytes)
: Small messages receive lower priorities than large messages, increasing the efficiency of the queue.

message class
: If the user includes a "Precedence:" field and value in a message, *sendmail* uses the value to select the message class from the configuration file. (Typical values might be "first-class" or "bulk.")

class work factor
: This factor is set in the configuration file with the **z** option; its default value is 1800.

number of recipients
: The number of recipients affects the load a message presents to the system. A message with a single recipient has a lower priority than one with a long recipient list.

recipient work factor
: This factor is set in the configuration file with the **y** option; its default value is 1000.

The priority algorithm is as follows:

```
priority=message_size-(message_class * z)+(num_recipients * y)
```

After assigning message priorities, *sendmail* orders the queue by using the following formula:

```
ordering = priority + creation_time
```

A message's priority can change each time an attempt is made to deliver it. The "work time factor" (set with the **Z** option) is a value that increments the priority, on the assumption that a message that has failed many times will tend to fail in the future.

Load Limiting

sendmail can queue (and not attempt to deliver) mail if the system load average exceeds a specified maximum. The *x* option defines this maximum limit. When the load average exceeds the maximum, *sendmail* tests each message's priority by using the following algorithm, where *q* is the value associated with the **q** option, and *x* is the value associated with the **x** option:

```
q / (load_average - x + 1)
```

In IRIX *sendmail,* the algorithm is modified slightly. The number of child processes forked by the *sendmail* daemon is added to the load average. Thus, a host with a load average of 1 but with 6 forked *sendmail* processes has an effective load average of 7.

After the final load average is calculated, *sendmail* compares it to the message priority for each message. If the priority is greater, *sendmail* sets the delivery mode to **q** (queue only).

The default value for the **q** option is 10000; each point of load average is worth 10000 priority points. The **X** option defines the load average at which *sendmail* refuses to accept network connections. Locally generated mail (including incoming UUCP mail) is still accepted.

Log Level

sendmail provides a comprehensive error- and event-logging capability. The **L** (log level) option in the configuration file determines the level of detail written to the log. The default value for the log level is 1; valid levels are as follows:

0	no logging
1	major problems only
2	message collections and failed deliveries
3	successful deliveries
4	messages being deferred (because a host is down, for example)
5	normal message queue-ups
6	unusual but benign incidents, such as trying to process a locked queue file
9	log internal queue ID to external message ID mappings; useful for tracing a message as it travels among several hosts
12	several messages of interest when debugging
16	verbose information regarding the queue

The Configuration File

This section begins with an overview of the configuration file, describes its semantics in detail, and includes hints on writing a customized version. Admittedly, the file's syntax is not easy to read or write. A more important design criterion is that the syntax be easy to parse, because it must be parsed each time *sendmail* starts up.

The Syntax

The configuration file is a series of lines, each of which begins with a character that defines the semantics for the rest of the line. A line beginning with a space or a tab is a continuation line (although the semantics are not well defined in many places). A blank line or a line beginning with the number symbol (#) is a comment.

Rewriting Rules—The S and R Commands

The rewriting rules are the core of address parsing. These rules form an ordered production system. During parsing, *sendmail* scans the set of rewriting rules, looking for a match on the left-hand side (ls) of the rule. When a rule matches, the address is replaced by the right-hand side (rhs) of the rule.

The rewriting rules are grouped into sets of one or more rules. Each set has an integer identifier called a *rule set number*. Each rule set acts like a subroutine: When the set is called, each rule is scanned in sequence until all rules have been scanned or until the rule set returns to the caller.

Some rule sets are used internally and have specific semantics. Others do not have specifically assigned semantics and can be referenced by mailer definitions or by other rule sets.

Each rule set begins with an **S** command, with the syntax

```
Sn
```

where *n* is the rule set identifier, an integer between 0 and 99. If the same rule set identifier is used more than once in a configuration file, the last definition is the one used.

Each line within the rule set begins with an **R** command and defines a rewrite rule. **R** commands have the syntax

```
Rlhs rhs [comments]
```

where the following conditions exist:

- The fields are separated by at least one tab character; embedded spaces with a field are acceptable.
- Any input matching the *lhs* pattern is rewritten according to the *rhs* pattern.
- Comments, if any, are ignored.

Define Macro—The D Command

The **D** command names a macro and defines its content. A macro name is a single ASCII character. *sendmail* defines several internal macros of its own. To avoid conflicts, use uppercase letters for all user-defined macros. Lowercase letters and special characters are reserved for system use. (A list of *sendmail*'s internally defined macros appears under the topic "Special Macros and Conditionals" on page 235.)

The Define Macro command takes one of the following forms:

`D`*xValue*

`D`*x*|*shell_command* `[`*arguments*`]`

where *x* is the name of the macro. The first form defines the macro to contain *Value.* The second form defines the macro as the first line seen on *stdout* of the specified shell command. The vertical bar must immediately follow the macro identifier; no white space is permitted. The remainder of the line is interpreted as arguments to the shell command.

Macros can be interpolated in most places by means of the escape sequence $x.

Caution: *sendmail* does not honor comments on macro definition lines. For example, the following line defines the D macro as "foo.com # my domain name":

```
DDfoo.com     # my domain name
```

Define Classes—The C and F Commands

The **C** command defines classes of words to match on the left-hand side of rewriting rules, where a "word" is a sequence of characters. A class name is a single uppercase letter. A class might define a site's local names and be used to eliminate attempts to send to oneself. Classes can be defined either directly in the configuration file or by being read in from a pipe or other file. The sequence cannot contain characters used as "operators" in addresses. (Operators are defined with the $o macro.)

The **C** command takes one of the following forms:

`C`*X word1 [word2 ...]*

`C`*X $x [$y ...]*

where *X* is the class name. The first form defines the class to match any of the named words. The second form reads the elements of the class from the expansion of the listed macros. The macros to be expanded must be leftmost in each token. Only one macro can be expanded per token. Any remainder in a token following a macro expansion is added as a separate token.

The **F** command defines a file from which to read the elements of a class, and takes either of the following forms:

`F`*X file [format]*

`F`*X | shell_command [arguments]*

The first form reads the elements of the class from the named file. If an optional format argument is present, it is passed as the control string to an **sscanf()** call and applied to each input line read from the named file, thus:

```
sscanf(const char *line, const char *format, char *new-element);
```

There must be no white space to the left of the filename.

The second form reads the elements of the class from *stdout* of the specified shell command. The entire contents of the line are taken to be a shell command followed by its arguments.

It is permissible to split class definitions across multiple lines. For example, these two forms are equivalent:

```
CHmonet picasso
```

and

```
CHmonet
CHpicasso
```

It is also permissible to define a class using both **C** and **F** commands. For example, the following commands define class H containing *monet*, *picasso*, the expansion of the $w macro, and all strings from the file */var/tmp/foofile* appearing before the first number symbol (#) on each line:

```
CHmonet picasso $w
FH/var/tmp/foofile %[^#]
```

Caution: *sendmail* does not honor comments that are not respected on class definition lines. For example, the following command defines the D class as containing "foo.com," "bar.com," "#," "my," "local," and "domains":

```
CD foo.com bar.com # my local domains
```

Define Mailer—The M Command

The **M** command defines programs and interfaces to mailers. The format is as follows:

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

where *name* is the name of the mailer (used internally only) and the *field=value* pairs define attributes of the mailer. The asterisk (*) indicates that the preceding bracketed structure may be repeated 0 or more times. That is, there may be multiple *field=value* pairs. The following list defines valid field names:

Path	The pathname of the mailer.
Flags	Special flags for this mailer; see "Flags, Options, and Files" on page 247 for a list of special flags.
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.

Define Header–The H Command

The **H** command defines the format of header lines that *sendmail* inserts into a message. The command format is as follows:

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

Continuation lines for an **H** command line appear in the outgoing message. *sendmail* macro-expands the *htemplate* before inserting it into the message. If *mflags* (which must be surrounded by question marks) appear in the command line, at least one of the specified flags must also appear in the mailer definition or the header will not appear in the message. However, if a header appears in the input message, the header also appears in the output, regardless of these flags.

Some headers have special semantics, which are described in "The Semantics" on page 234.

Set Option—The O Command

Several *sendmail* options can be set by a command line in the configuration file. Each option is identified by a single character.

The format of the **O** command line is as follows:

O*o value*

The command sets option *o* to *value.* Depending on the option, *value* is a string; an integer; a Boolean (with legal values "t," "T," "f," or "F" and a default of TRUE); or a time interval.

The **K** command is a new command available in IRIX *sendmail.* This command defines (K)eyed databases that are accessible by means of the lookup operators. See "Define Keyed Files—The K Command" on page 234 for a complete description of the **K** command.

Define Trusted Users—The T Command

Trusted users are permitted to override the sender address by using the *-f* flag. Typically, trusted users are limited to *root*, *uucp*, and *network.* On some systems it may be convenient to include other users. For example, there may be a separate *uucp* login for each host. Note, though, that the concept of trusted users in *sendmail* bears no relation to Trusted IRIX/B, or to the concept of trusted (secure) operating systems in general.

The format of the **T** command is as follows:

T*user1 user2* . . .

A configuration file can contain multiple **T** lines.

Define Precedence—The P Command

The **P** command defines values for the Precedence field. The format of the command line is as follows:

`P`*name*=*num*

When *sendmail* matches the value in a message's Precedence field with the *name* in a **P** command line, *sendmail* sets the message's class to *num.* A higher number indicates a higher precedence. Negative numbers indicate that error messages will not be returned to the sender. The default precedence is zero. For example, a list of precedences might be

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

Define Keyed Files—The K Command

The **K** command defines a symbolic name for accessing a database. The format of the command line is as follows:

`K`*name type file*

The *name* value is the name used to specify the database in a lookup command. The *type* defines the type of database file, which can be one of the following:

dbm	Support for the ndbm(3) library.
nis	Support for NIS (YP) maps. NIS+ is not supported in this version.
host	Support for DNS lookups.
dequote	A "pseudo-map" (that is, one that does not have any external data) that allows a configuration file to break apart a quoted string in the address. This is primarily useful for DECnet addresses, which often have quoted addresses that need to be unwrapped on gateways.

The *file* value specifies the filename of the database to be searched.

The Semantics

This section describes the semantics of the configuration file, including special macros, conditionals, special classes, and the "error" mailer.

Special Macros and Conditionals

Macros are interpolated by means of the construct $*x*, where *x* is the name of the macro to be interpolated. Special macros are named with lowercase letters; they either have special semantics or pass information into or out of *sendmail.* Some special characters are also reserved, and are used to provide conditionals and other functions.

The following syntax specifies a conditional:

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

This example interpolates *text1* if the macro *$x* is set, and *text2* otherwise.

The "else" (*$*|) clause can be omitted.

The following macros *must b*e defined if information is to be transmitted into *sendmail*:

e	SMTP entry message, which prints out when STMP starts up
j	"official" domain name for this site; must be the first word of the *$e* macro
l	format of the UNIX "From" line; usually a constant
n	name of the daemon (for error messages); usually a constant
o	set of token operators in addresses
q	default format of sender address

The $o macro consists of a list of characters that are treated as tokens themselves and serve to separate tokens during parsing. For example, if @ were in the $o macro, then the input string "a@b" would scan as three tokens: a, @, and b.

Here are several examples of these macro definitions:

```
De$j Sendmail $v/$Z ready at $b

Dj$w

DlFrom $g  $d

DnMAILER-DAEMON

Do.:%@!^=/[]

Dq$g$?x ($x)$.
```

An acceptable alternative format for the *$q* macro is "$?x$x $.<$g>" . This syntax corresponds to the following two formats:

```
jd@company.com (John Doe)
```

```
John Doe <jd@company.com>
```

Some macros are defined by *sendmail* for interpolation into arguments passed to mailers or for other contexts. These macros are:

a	origination date in RFC 822 format
b	current date in RFC 822 format
c	hop count
d	date in UNIX (ctime) format
f	sender ("From") address
g	sender address relative to the recipient
h	recipient host
i	queue ID
p	PID for *sendmail*
r	protocol used
s	sender's hostname
t	a numeric representation of the current time
u	recipient user
v	version number of *sendmail*
w	hostname of this site
x	full name of the sender
z	home directory of the recipient

Macros $a, $b, and $d specify three dates that can be used. The $a and $b macros are in RFC 822 format. $a is the time extracted from the "Date:" line of the message; $b is the current date and time (used for postmarks). If no "Date:" line appears in the incoming message, $a is also set to the current time. The $d macro is equivalent to the $a macro in UNIX format (as described on the ctime(3C) reference page).

The $c macro is set to the "hop count," the number of times this message has been processed. It can be determined by the **-h** flag on the command line or by counting the time stamps in the message.

The $f macro is the ID of the sender as originally determined; when a message is sent to a specific host, the $g macro is set to the address of the sender *relative to the recipient.* For example, if *jd* sends to *buddy@USomewhere.edu* from the machine *company.com,* the $f macro will be *jd* and the $g macro will be *jd@company.com.*

When a message is sent, the $h, $u, and $z macros are 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 $i macro is set to the queue ID on this host; when included in the time-stamp line, it can be extremely useful for tracking messages.

The $p and $t macros are used to create unique strings (for example, for the Message-Id field). The $r and $s macros are set to the protocol used to communicate with *sendmail* and the name of the sending host; these macros are not supported in the current version.

The $v macro is set to be the version number of *sendmail,* which normally appears in time stamps and is extremely useful for debugging. The $w macro is set to the name of this host, if it can be determined.

The $x macro is set to the full name of the sender, derived from one of these sources (in this order):

- a flag to *sendmail*
- the value of the "Full-name:" line in the header if it exists
- the comment field of a "From:" line
- for messages originating locally, the value from the */etc/passwd* file

Special Classes

The w class is the set of all names this host is known by, and can be used to match local hostnames.

The Left-Hand Side

The left-hand side of a rewriting rule contains a pattern. Words are simply matched directly. A dollar sign introduces the following meta-syntax:

$*	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 a match occurs, it is 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

```
jd@company.com
```

the rule will match, and the values passed to the rhs will be

$1	jd
$2	company.com

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. Right-hand-side tokens are inserted exactly as they appear unless they begin with a dollar sign. The meta-syntax is:

$*n*	Substitute indefinite token *n* from lhs; substitute the corresponding value from a $+, $-, $*, $=, or $~ match on the lhs; can be used anywhere.
$[*hostname*$]	Canonicalize *hostname*; a hostname enclosed between $[and $] is looked up by the **gethostbyname()** routines and replaced by the canonical name. For example, $[frodo$] might become *frodo.fantasy.com* and $[[192.48.153.1]$] would become *sgi.com*. If **gethostbyname()** encounters an error, *hostname* will be returned unchanged.
$[*name*$:*default*$]	This is an extended version of the preceding construct, and provides a way to determine whether the canonicalization was successful. Using this syntax, the *default* is returned if the canonicalization step fails. For example, $[frodo$:FAIL$] becomes FAIL if **gethostbyname()** fails to canonicalize *frodo*.

$(*x key*$@*arg*$:*default*$)

Look up key in DBM or NIS database *x*. This is the database lookup syntax; *x* corresponds to a database previously declared using the *K* command (described in "Define Keyed Files—The K Command" on page 234) and *key* is the string that should be searched for in the database. The *arg* and *default* arguments are optional. The *default* is returned if the *key* was not found in the database. If neither the *default* nor a matching *key* is found, the whole expression expands to the value of *key*. However, if a result is found, it is used as the format string of a **sprintf()** expression, with the *arg* as extra argument. Thus, database values with "%s" strings embedded in them can be useful when rewriting expressions. These values could typically be used with the *pathalias* program to expand routes without leaving *sendmail.*

${*x query*$:*default*$}

Look up query in DNS database *x*. This is the DNS database lookup syntax. The various values of *x* are internally defined and correspond to various DNS databases. The *default* argument is optional and will be returned if *query* cannot be found in the specified database.

The values for *x* are listed below.

@	Return MX record for query.
.	As above, but use domain search rules.
:	Return MR record for *query.*
?	Return MB record for *query.*
c	Expand *query* to its canonical name following MX records.
C	As above, but use domain search rules.
h	Like *${c* but for A records.
H	As above, but use domain search rules.

When domain search rules are requested, *sendmail* sets the RES_DNSRCH flag when calling the resolver. See the resolver(4) reference page for further information.

$>*n*

Call rule set *n*; causes the remainder of the line to be substituted as usual and then passed as the argument to rule set *n*. The final value of rule set *n* becomes the substitution for this construct.

Multiple calls can be embedded on the rhs. For example, >32>33$1 would make the substitution for $1 and pass the result to rule set 33. The result from rule set 33 would then be passed as the input to rule set 32. Finally, the entire construct would be replaced with the result from rule set 32.

Only embedded rule set calls in the form outlined above are supported. Rule sets calls cannot be arbitrarily placed within the rhs.

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

Resolve to mailer; the $# syntax should be used only in rule set 0. The syntax causes evaluation of the rule set to terminate immediately and signals sendmail that the address has completely resolved. This process specifies the {*mailer*, *host*, *user*} triple 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 a multi-part value.

An entire rhs may also be prefixed by a $@ or a $: to control evaluation.

The $@ prefix causes the rule set to return with the remainder of the rhs as the value. The $: prefix causes the rule to terminate immediately, but the rule set 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 $> specification. For example,

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

matches anything, passes that to rule set 7, and continues; the $: is necessary to avoid an infinite loop.

Substitution occurs in the order described: Parameters from the lhs are substituted, host names are canonicalized, “subroutines” are called, and finally $#, $@, and $: are processed.

Semantics of Rewriting Rule Sets

There are five rewriting rule sets that have specific semantics. Figure B-1 shows the relationship among these rule sets.

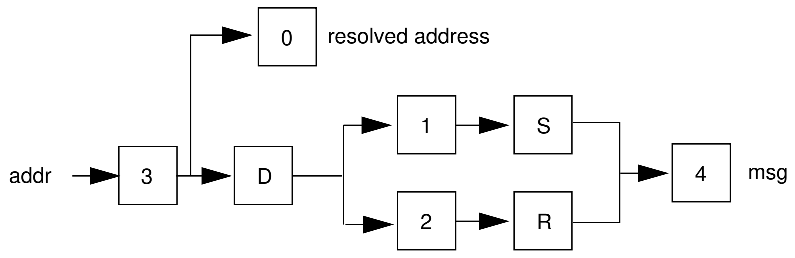

Figure B-1 Semantics of Rewriting Rule Sets

Rule set 3 should turn the address into canonical form. This form should have the following basic syntax:

```
local-part@host-domain-spec
```

If no "at" (@) sign is specified, then the host-domain-spec can be appended from the sender address (if the C flag is set in the mailer definition corresponding to the *sending* mailer). *sendmail* applies rule set 3 before doing anything with any address.

Next, rule set 0 is applied to an address that actually specifies recipients. The address 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.

Rule sets 1 and 2 are applied to all sender and recipient addresses, respectively. They are applied before any specification in the mailer definition. They must never resolve.

Rule set 4 is applied to all addresses in the message. It is typically used to translate from internal to external form.

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 following entry on the rhs of a rule causes the specified error to be generated if the lhs match:

```
$#error$:Host unknown in this domain
```

This mailer is functional only in rule set 0.

Relevant Issues

This section discusses testing and debugging the rewrite rules and building mailer definitions.

Testing and Debugging the Rewrite Rules

As part of building or modifying a configuration file, you should test the new file. *sendmail* provides a number of built-in tools to assist in this task. The subsections that follow discuss tools and techniques for debugging the rewrite rules.

Using Alternative Configuration Files

Using the *-C* command-line flag causes *sendmail* to read an alternate configuration file. This feature is helpful during debugging because it permits modifications and testing on a separate copy of the configuration file from the one currently in use. This precaution eliminates the chance that a buggy configuration file will be used by an instance of *sendmail* that is trying to deliver real mail. This flag also provides a convenient way to test any number of configuration files without fussy and potentially confusing renames. See "Using a Different Configuration File" on page 224 and "Command-Line Flags" on page 247 for more information.

Test Mode

Invoking *sendmail* with the *-bt* flag causes it to run in "test mode." For example, the following command invokes *sendmail* in test mode and causes it to read configuration file *test.cf:*

```
/usr/lib/sendmail –bt –Ctest.cf
```

In this mode, *sendmail* processes lines of the form

rwsets address

where *rwsets* is the list of rewriting sets to use and *address* is an address to which you apply the sets. In test mode, *sendmail* shows the steps it takes as it proceeds, finally showing the final address.

A comma-separated list of rule sets causes sequential application of rules to an input. For example, the following command first applies rule set 3 to the value *monet@giverny*. Rule set 1 is applied to the output of rule set 3, followed similarly by rule sets 21 and 4:

```
3,1,21,4 monet@giverny
```

Note: Some versions of *sendmail*, including those provided with all versions of IRIX prior to IRIX 4.0, automatically apply rule set 3 to input before applying the requested rule set sequence. Versions of *sendmail* in IRIX 4.0 and later do *not* apply rule set 3; rule set 3 must be specifically requested.

The input and output of each rule set is displayed. For example, input of

```
3,0 foo@bar
```

might result in output that looks like this:

```
rewrite: ruleset 3 input: foo @ bar
rewrite: ruleset 3 returns: foo < @ bar >
rewrite: ruleset 0 input: foo < @ bar >
rewrite: ruleset 30 input: foo < @ bar . com >
rewrite: ruleset 30 returns: foo < @ bar . com >
rewrite: ruleset 0 returns:
$# forgn $@ bar . com $: foo < @ bar . com >
```

This output indicates that, given the address *foo@bar*, rule set 0 will select the *forgn* mailer and direct it to connect to host *bar.com*, which will be told to send the mail on to *foo@bar.com*. Furthermore, rule set 0 “called” rule set 30 at one point while processing the address.

The -d21 Debugging Flag

The **-d21** debugging flag causes *sendmail* to display detailed information about the rewrite process. This flag is most useful when used with the test mode described in the preceding subsection. The most useful setting of this flag is **-d21.12**, which shows all rewrite steps. Higher levels of the **-d21** flag are rarely needed and create enormous amounts of output.

The Debugging Rewrite Rule

The standard */usr/lib/sendmail.cf* file supplied with IRIX includes a special “debugging” rewrite rule. This rule is defined as follows:

```
# insert this handy debugging line wherever you have problems
#R$*            $:$>99$1
```

Note that rule set 99 is an empty rule set that does nothing. Placing one or more (uncommented) copies of this rule anywhere within a rule set forces *sendmail* to display an intermediate rewrite result without using the *-d21* flag. The following test mode output illustrates the use of the debugging rewrite rule:

```
rewrite: ruleset 3 input: foo @ bar
rewrite: ruleset 3 returns: foo < @ bar >
rewrite: ruleset 0 input: foo < @ bar >
rewrite: ruleset 99 input: foo < @ bar . com >
rewrite: ruleset 99 returns: foo < @ bar . com >
rewrite: ruleset 99 input: foo < @ barcom >
rewrite: ruleset 99 returns: foo < @ barcom >
rewrite: ruleset 0 returns:
$# ether $@ barcom $: foo < @ barcom >
```

Note that somewhere between the first and second appearance of the debugging rewrite rule in rule set 0, the host name was mangled from *bar.com* to *barcom.*

Building Mailer Definitions

To add an outgoing mailer to a mail system, you must define the characteristics of the mailer. Each mailer must have an internal name. This name 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 is accessed by means of an IPC connection (socket), use the string “[IPC]” instead.

The *F* field defines the mailer flags. Specify an *f* or *r* flag to pass the name of the sender as an **-f** or **-r** flag respectively. These flags are passed only if they were passed to *sendmail,* so that mailers that give errors under some circumstances can be placated. If the mailer is not picky, just specify “-f $g” in the *argv* template. If the mailer must be called as *root*, use the *S* flag; this flag will not reset the user ID before calling the mailer. (*sendmail* must be running *setuid* to root for this technique to work.)

If this mailer is local (that is, it will perform final delivery rather than another network hop), use the *l* flag. Quote characters (backslashes and quotation marks) can be stripped from addresses if the *s* flag is specified; if it is not, they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction, use the *m* flag. If this flag is on, the *argv* template containing $u will be repeated for each unique user on a given host. The *e* flag marks the mailer as being expensive, causing *sendmail* to defer connection until a queue run. (For this technique to be effective, you must use the *c* configuration option.)

An unusual case is the *C* flag: it applies to the mailer the message is received from, rather than the mailer being sent to. If this flag is set, the domain specification of the sender (that is, the *@host.domain* part) is saved and is appended to any addresses in the message that do not already contain a domain specification. For example, if the *C* flag is defined in the mailer corresponding to *jd@company.com,* a message of the form

```
From: jd@company.com
To: buddy@USomewhere.edu, jane
```

will be modified to

```
From: jd@company.com
To: buddy@USomewhere.edu, jane@company.com
```

Other flags are described in “Mailer Flags” on page 251.

The *S* and *R* fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses, respectively. These sets are applied after the sending domain is appended and the general rewriting sets (1 and 2) are applied, but before the output rewrite (rule set 4) is applied. A typical usage is to append the current domain to addresses that do not already have a domain.

For example, depending on the domain it is being shipped into, a header of the form “From: jd” might be changed to “From: jd@company.com” or “From: company!jd.”

These sets can also be used to do special-purpose output rewriting with rule set 4.

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

Finally, an *argv* template is given as the *E* field. It can 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 ]
```

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

For example, the following specification specifies a mailer to do local delivery and a mailer for Ethernet delivery:

```
Mlocal, P=/bin/mail, F=EDFMlsmhu, S=10, R=20, A=mail -s -d $u
Mether, P=[IPC], F=mDFMhuXC, S=11, R=21, M=1000000, E=\r\n, \
    A=IPC $h
```

The first mailer is called "local" and is located in the file */bin/mail.* It has the following characteristics:

- It escapes lines beginning with "From" in the message with a ">" sign.
- It expects "Date:", "From:", and "Message-Id:" header lines.
- It does local delivery.
- It strips quotation marks from addresses.
- It sends to multiple users at once.
- It expects uppercase to be preserved in both host and user names.

Rule set 10 is to be applied to sender addresses in the message and rule set 20 is to be applied to recipient addresses. The *argv* to send to a message is the word *mail,* the word *-s,* the word *-d,* and words containing the name of the receiving user.

The second mailer is called "ether" and is connected with an IPC connection. It has the following characteristics:

- It handles multiple users at the same time.
- It expects "Date:", "From:", and "Message-Id:" header lines.
- It expects uppercase to be preserved in both host and user names.
- It uses the hidden-dot algorithm of RFC 821.
- It rewrites addresses so that any domain from the sender address is appended to any receiver name without a domain.

Sender addresses are processed by rule set 11; recipient addresses, by rule set 21. There is a 1,000,000-byte limit on messages passed through this mailer. The EOL string for this mailer is "\r\n" and the argument passed to the mailer is the name of the recipient host.

Flags, Options, and Files

This section contains information on the following topics:

- command-line flags
- configuration options
- mailer flags
- summary of support files
- debugging flags

Command-Line Flags

Flags must precede addresses. The flags are:

-b*x* Set operation mode to *x*. Operation modes are:

a	Run in ARPANET mode. The special processing for the ARPANET includes reading the “From:” and “Sender:” lines from the header to find the sender, printing ARPANET-style messages (preceded by three-digit reply codes for compatibility with the FTP protocol) and ending lines of error messages with <CRLF>.
d	Run as a daemon.
i	Initialize the alias database.
m	Deliver mail in the usual way (default).
p	Print the mail queue.
s	Speak SMTP on input side.
t	Run in test mode.
v	Just verify addresses, don’t collect or deliver.
z	Freeze the configuration file.

-C*file* Use a different configuration file. *sendmail* runs as the invoking user (rather than root) when this flag is specified.

-d*flag* [-*flag*] [.*level*]	Set debugging flag (or range of flags) to the specified level. (The default is 1.) See "Debugging Flags" on page 254.
-F*name*	Set the full name of the sender to *name.*
-f*name*	Set the name of the "From" person (the sender of the mail). This flag is ignored unless the user appears in the list of "trusted" users, or *name* is the same as the user's name.
-h*cnt*	Set the "hop count" to *cnt*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.
-n	Don't do aliasing.
-o*x value*	Set configuration option *x* to the specified *value.* These options are described in section "Configuration Options" on page 248.
-q[*time*]	Process the queued mail. If the time is given, *sendmail* will run through the queue at the specified interval to deliver queued mail; otherwise, it runs only once. See "Queue Mode" on page 222.
-r *name*	An alternative and obsolete form of *-f.*
-t	Read the header for "To:", "Cc:", and "Bcc:" lines, and send the message to everyone listed in those lists. The "Bcc:" line is deleted before sending. Any addresses in the argument vector are deleted from the send list.
-v	Go into verbose mode: Alias expansions are announced, and so on.

Configuration Options

You can set the following options by using the *-o* flag on the command line or the *O* line in the configuration file. Many of these options 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 alias 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 the entry 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 option requires queueing.	
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	Rebuild the alias database if necessary and possible. If this option is not set, *sendmail* will not rebuild the alias database until you explicitly request it to do so (by using *sendmail -bi*).
e*x*	Handle errors by using mode *x.* The values for *x* are:	
	p	Print error messages (default).
	q	Print no messages; just give exit status.
	m	Mail back errors.
	w	Write back errors (mail the errors if user not logged in).
	e	Mail back errors and always give zero exit status.
F*mode*	Set the UNIX file mode to use when creating queue files and "frozen configuration" files.	
f	Save UNIX style "From" lines at the front of headers. Normally these lines are assumed to be redundant and are discarded.	
g*n*	Set the default group ID for running mailers to *n*.	
H*file*	Specify the help file for SMTP.	
I	Insist that the BIND name server be running to resolve hostnames and MX records. Treat ECONNREFUSED errors from the resolver as temporary failures. In general, you should set this option only if you are running the name server. Set this option if the */etc/hosts* file does not include all known hosts or if you are using the MX (mail forwarding) feature of the BIND name server. The name server is still consulted even if this option is not set, but *sendmail* resorts to reading */etc/hosts* if the name server is not available.	
i	Do not interpret dots on a line by themselves as a message terminator.	

K*timeout*	Define the maximum amount of time a cached connection is permitted to idle without activity. The *timeout* is given as a tagged number, with “s” for seconds, “m” minutes, “h” hours, “d” days, and “w” weeks. For example, “K1h30m” and “K90m” both set the *timeout* to one hour thirty minutes.
L*n*	Set the log level to *n.*
M*x value*	Set the macro *x* to *value.* This option can be used only from the command line.
m	Send to “me” (the sender) even if the sender is in an alias expansion.
N*netname*	Set the name of the home (local) network. If the name of a connecting host (determined by a call to **gethostbyaddr()**) is unqualified (contains no dots), a single dot and *netname* will be appended to *sendmail*’s idea of the name of the connecting host. Later, the argument of the SMTP “HELO” command from the connecting host will be checked against the name of the connecting host as determined above. If they do not match, “Received:” lines are augmented by the connecting hostname that *sendmail* has generated so that messages can be traced accurately.
n	Validate the right-hand side when building the alias database.
o	Assume that the headers may be in an old format, in which spaces delimit names. This option 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 written with commas between the names.
P*addr*	Add “postmaster” address *addr* to the “Cc:” list of all error messages.
Q*dir*	Use dir as the queue directory.
q*factor*	Use factor as the multiplier in the function to decide when to queue messages rather than attempting to send them. This value is divided by the difference between the current load average and the load average limit (*x* option) to determine the maximum message priority that will be sent. This value defaults to 10000.
r*time*	Cause a timeout on reads after *time* interval.
S*file*	Log statistics in the named *file.*

s	Be super-safe when running; that is, always instantiate the queue file, even if attempting immediate delivery. *sendmail* always instantiates the queue file before returning control to the client under any circumstances.
T*time*	Set the queue timeout to *time.* After this interval, messages that have not been successfully sent are returned to the sender.
u*n*	Set the default user ID for mailers to *n*. Mailers without the S flag in the mailer definition run as this user.
v	Run in verbose mode.
x*LA*	Use *LA* as the system load average limit when deciding whether to queue messages rather than attempting to send them.
X*LA*	When the system load average exceeds LA, refuse incoming SMTP connections.
y*factor*	This factor is multiplied by the number of recipients and added to the priority. Therefore, this value penalizes messages with large numbers of recipients.
Y	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*factor*	This factor is multiplied by the message class (determined by the Precedence field in the user header and the precedence declaration lines in the configuration file) and subtracted from the priority. Therefore, messages with a higher class are favored.
Z*factor*	This factor is added to the priority every time a message is processed. Therefore, this value penalizes messages that are processed frequently.

Mailer Flags

The following flags can be set in the F field of a mailer definition in the *sendmail.cf* file:

B	Don’t wait for SMTP responses.
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 rule set 3 have the “@domain” clause from the sender appended. This flag allows mail with headers of the form

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

to be rewritten automatically as

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

D	This mailer expects a “Date:” header line.
E	Escape any lines beginning with “From” in the message with a “>” sign.
e	This mailer is expensive to connect to; usually, avoid connecting. Any necessary connection occurs during a queue run.
F	The mailer expects a “From:” header line.
f	The mailer expects a *-f from* flag, but only if this is a network forwarding operation. (That is, the mailer will give an error if the executing user does not have special permissions.)
h	Uppercase should be preserved in hostnames for this mailer.
I	This mailer can use certain special SMTP features when transferring mail to another system running *sendmail*. This option is not required; if it is omitted, the transmission still operates successfully, although perhaps not as efficiently as possible.
L	Limit the line lengths as specified in RFC 821.
l	This mailer is local; final delivery will be performed.
M	This mailer expects a “Message-Id:” header line.
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.
n	Do not insert a UNIX style “From” line on the front of the message.
P	This mailer expects a “Return-Path:” line.
p	Use the return path in the SMTP “MAIL FROM:” command rather than just the return address; although this usage is required in RFC 821, many hosts do not process return paths properly.
r	Same as *f*, but sends an *-r* flag.

S	Don’t reset the user ID before calling the mailer. Used in a secure environment where *sendmail* runs as root, this option could avoid forged addresses. This flag is suppressed if given from an “unsafe” environment (such as a user’s *mail.cf* file).
s	Strip quotation characters from the address before calling the mailer.
U	This mailer wants UNIX style “From” lines with the UUCP-style “remote from <host>” on the end.
u	Preserve uppercase characters in user names for this mailer.
V	Make all addresses UUCP !-relative to recipient or sender nodes. Addresses of the form *recipient_host!foo!bar* are rewritten as *foo!bar*. Addresses of the form *mumble!grumble* are rewritten as *sender_host!mumble!grumble*.
X	This mailer expects to use the hidden-dot algorithm as specified in RFC 821; basically, any line beginning with a dot has an extra dot prepended (to be stripped at the other end). This action ensures that a line containing a dot will not terminate the message prematurely.
x	This mailer expects a “Full-Name:” header line.

Support Files

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

/usr/lib/sendmail
: The sendmail program.

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

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

/usr/lib/sendmail.fc
: The configuration file represented as a memory image (the “frozen configuration”).

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

/usr/lib/sendmail.st	A statistics file; need not be present.
/usr/lib/sendmail.killed	A text file that contains the names of all known “dead” hosts (hosts that no longer exist or cannot receive mail for some reason).
/usr/lib/aliases	The text version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in *ndbm* format.
/var/spool/mqueue	The directory in which the mail queue and temporary files reside.
*/var/spool/mqueue/qf**	Control (queue) files for messages.
*/var/spool/mqueue/df**	Data files.
*/var/spool/mqueue/tf**	Temporary versions of the *qf* files, used during queue-file rebuild.
*/var/spool/mqueue/nf**	A file used when a unique ID is created.
*/var/spool/mqueue/xf**	A transcript of the current session.
/usr/bin/mailq	Prints a listing of the mail queue; using this file is equivalent to using the *-bp* flag to *sendmail.*
/etc/init.d/mail	Shell script for starting and stopping the *sendmail* daemon.
/bin/mail	Program that *sendmail* uses as the “local” mailer.

Debugging Flags

The following list includes all known debugging flags. Flags that are especially useful are marked with an asterisk (*).

0.1*	Force daemon to run in foreground.
0.4*	Show known names for local host.
0.15	Print configuration file.

0.44	Have **printav()** print addresses of elements.
1.1*	Show mail "From" address for locally generated mail.
2.1*	Print exit status and envelope flags.
5.4	Print arguments to **tick()** calls.
5.5	Print arguments to **setevent()** and **clrevent()** calls.
5.6	Print event queue on **tick()** call.
6.1	Indicate call to **savemail()** or **returntosender()** error processing.
6.5	Trace states in **savemail()** state machine.
7.1*	Print information on envelope assigned to queue file.
7.2*	Print selected queue-file name.
7.20*	Print intermediate queue-file name selections.
8.1*	Print various information about resolver calls.
9.1*	Show results from **gethostbyaddr()** call.
10.1*	Print message delivery information.
11.1	Indicate call to **openmailer()**.
12.1*	Display **remotename()** input and output.
13.1	**sendall()**—print addresses being sent to
13.3	**sendall()**—print each address in loop looking for failure.
13.4	**sendall()**—print who gets the error.
14.2	Indicate **commaize()** calls.
15.1	Indicate port or socket number used by **getrequests()**.
15.2	Indicate when **getrequests()** forks or returns.
15.15	Set DEBUG socket option in **getrequests()**.
16.1*	Indicate host, address, and socket being connected to in **makeconnection()**.
16.14	Set DEBUG socket option in **makeconnection()**.
18.1*	Show SMTP chatter.
18.100	Suspend *sendmail* after reading each SMTP reply.

20.1*	Display **parseaddr()** input and output.
21.2*	Show rewrite rule-set subroutine calls/returns and input/output, and display run-time macro expansions.
21.3*	Indicate rewrite subroutine call from inside rewrite rule.
21.4*	Display rewrite results.
21.10*	Indicate rule failures.
21.12*	Indicate rule matches and display address-rewrite steps.
21.15*	Show rewrite substitutions.
21.35	Display elements in pattern and subject.
22.36	Display **prescan()** processing.
22.45	Display more **prescan()** processing.
22.101	Display even more **prescan()** processing.
25.1*	Show “To” list designations.
26.1*	Show recipient designations/duplicate suppression.
26.6*	Show recipient password-match processing.
27.1*	Print alias and forward transformations and errors.
27.3	Print detailed **aliaslookup()** information.
30.1	Indicate end of headers when collecting a message.
30.2	Print arguments to **eatfrom()** calls.
30.3	Indicate when adding an “Apparently-To” header to the message.
31.6	Indicate call to **chompheader()** and header to be processed.
32.1	Display collected header.
33.1	Display **crackaddr()** input/output.
35.9*	Display macro definitions.
35.24	Display macro expansions.
36.5	Show symbol table processing.
36.9	Show symbol table hash function result.
37.1*	Display options as set.

37.2*	Show rewrite class loading.
40.1*	Indicate queueing of messages and display queue contents.
40.4*	Display queue control file contents.
40.5*	Display information about message-controlling user.
41.2	Indicate **orderq()** failure to open control file.
45.1	Indicate **setsender()** calls.
50.1	Indicate **dropenvelope()** calls.
51.4	Don’t remove transcript files (*qxAAXXXXX* files).
52.1	Indicate call to **disconnect()**; print I/O file descriptors.
52.5	Don’t perform disconnect.
60.1*	Print information about alias database accesses.
61.1*	Print information about MX record lookups.