Mail Reference Manual

IRIS-4D Series

SiliconGraphics
Computer Systems

Document number: 007-0880-010

Mail Reference Manual

Version 1.0

Document Number 007-0880-010

Technical Publications:

Gail Kesner
Amy B. W. Smith
Diane Wilford

Engineering:

Vernon Schryver

Mail Reference Manual
Version 1.0
Document Number 007-0880-010

Silicon Graphics, Inc.
Mountain View, California

Introduction

Network mail is supported on the IRIS-4D Series workstation. The *Mail Reference Manual* is a collection of reference materials on network mail. The manuals included here provide in-depth information on the mechanisms of network mail.

Read this manual if you need information about tailoring your mail environment. If you need information about sending mail or a brief discussion about configuring your IRIS-4D Series workstation for mail, see the chapter on Mail in the *IRIS-4D Series Communications Guide.*

The following documents are included:

- *Mail Systems and Addressing in 4.2bsd*
- *Sendmail Installation and Operation Guide*
- *Sendmail—An Internetwork Mail Router*
- *The Domain Naming Convention for Internet User Applications*

The first three documents were written by Eric Allman who has done research in routing and internetwork mail. The last document is a Request For Comment (RFC819) written by Zaw-Zing Su and Jon Postel.

Mail Systems

Mail Systems and Addressing in 4.2bsd MS-1

1. **Definitions** MS-5
   1.1 User and Identification MS-5
   1.2 Address MS-6
   1.3 Route MS-6

2. **Design Goals** MS-7

3. **Usage** MS-11
   3.1 Address Formats MS-11
   3.2 Mail to Files and Programs MS-12
   3.3 Aliasing, Forwarding, Inclusion MS-13
      3.3.1 Aliasing MS-13
      3.3.2 Forwarding MS-13
      3.3.3 Inclusion MS-14
   3.4 Message Collection MS-14

4. **The UUCP Problem** MS-15
   4.1 Defining the Namespace MS-16
   4.2 Creating and Propagating the Software MS-16
   4.3 Building and Maintaining the Database MS-17
   4.4 Logical Structure MS-18

5. **Comparison with Delivermail** MS-19

Mail Systems and Addressing in 4.2bsd

Eric Allman[†]

Britton-Lee, Inc.
1919 Addison Street, Suite 105.
Berkeley, California 94704

Abstract

Routing mail through a heterogeneous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an ad hoc basis. However, this approach has become unmanageable as internets grow.

Sendmail acts a unified ''post office'' to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both old and new format addresses. The new format is ''domain-based'', a flexible technique that can handle many common situations. Sendmail is not intended to perform user interface functions.

Sendmail will replace deliver mail in the Berkeley 4.2 distribution. Several major hosts are now or will soon be running sendmail. This change will affect any users that route mail through a sendmail gateway. The changes that will be user visible are emphasized.

The mail system to appear in 4.2bsd will contain a number of changes. Most of these changes are based on the replacement of *delivermail* with a new module called *sendmail*. *Sendmail* implements a general internetwork

† A considerable part of this work was done while under the employ of the INGRES Project at the University of California at Berkeley.

mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration. Of key interest to the mail system user will be the changes in the network addressing structure.

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characteristics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require that the route the message takes be explicitly specified by the sender, simplifying the database update problem since only adjacent hosts must be entered into the system tables, while others use logical addressing, where the sender specifies the location of the recipient but not how to get there. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

Internet standards seek to eliminate these problems. Initially, these proposed expanding the address pairs to address triples, consisting of {network, host, username} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was changed to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root. The domain technique separates the issue of physical versus logical addressing. For example, an address of the form ''eric@a.cc.berkeley.arpa'' describes the logical organization of the address space (user ''eric'' on host ''a'' in the Computer Center at Berkeley) but not the physical networks used (for example, this could go over different networks depending on whether ''a'' were on an ethernet or a store-and-forward network).

Sendmail is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, *sendmail* is designed to assist a graceful transition to consistent internetwork addressing schemes.

Section 1 defines some of the terms frequently left fuzzy when working in mail systems. Section 2 discusses the design goals for *sendmail*. In section 3, the new address formats and basic features of *sendmail* are described. Section 4 discusses some of the special problems of the UUCP network. The differences between *sendmail* and *delivermail* are presented in section 5.

> **DISCLAIMER:** A number of examples in this paper use names of actual people and organizations. This is not intended to imply a commitment or even an intellectual agreement on the part of these people or organizations. In particular, Bell Telephone Laboratories (BTL), Digital Equipment Corporation (DEC), Lawrence Berkeley Laboratories (LBL), Britton-Lee Incorporated (BLI), and the University of California at Berkeley are not committed to any of these proposals at this time. Much of this paper represents no more than the personal opinions of the author.

1. Definitions

There are four basic concepts that must be clearly distinguished when dealing with mail systems: the user (or the user's agent), the user's identification, the user's address, and the route. These are distinguished primarily by their position independence.

1.1 User and Identification

The user is the being (a person or program) that is creating or receiving a message. An *agent* is an entity operating on behalf of the user — such as a secretary who handles my mail. or a program that automatically returns a message such as ''I am at the UNICOM conference.''

The identification is the tag that goes along with the particular user. This tag is completely independent of location. For example, my identification is the string ''Eric Allman'', and this identification does not change whether I am located at U.C. Berkeley, at Britton-Lee, or at a scientific institute in Austria.

Since the identification is frequently ambiguous (e.g., there are two ''Robert Henry''s at Berkeley) it is common to add other disambiguating information that is not strictly part of the identification (e.g., Robert ''Code Generator'' Henry versus Robert ''System Administrator'' Henry).

1.2 Address

The address specifies a location. As I move around, my address changes. For example, my address might change from *eric@Berkeley.ARPA* to *eric@bli.UUCP* or *allman@IIASA.Austria* depending on my current affiliation.

However, an address is independent of the location of anyone else. That is, my address remains the same to everyone who might be sending me mail. For example, a person at MIT and a person at USC could both send to *eric@Berkeley.ARPA* and have it arrive to the same mailbox.

Ideally a ''white pages'' service would be provided to map user identifications into addresses (for example, see [Solomon81]). Currently this is handled by passing around scraps of paper or by calling people on the telephone to find out their address.

1.3 Route

While an address specifies *where* to find a mailbox, a route specifies *how* to find the mailbox. Specifically, it specifies a path from sender to receiver. As such, the route is potentially different for every pair of people in the electronic universe.

Normally the route is hidden from the user by the software. However, some networks put the burden of determining the route onto the sender. Although this simplifies the software, it also greatly impairs the usability for most users. The UUCP network is an example of such a network.

2. Design Goals

Design goals for *sendmail*[1] include:

1. Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail, Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78]. ARPANET mail [Crocker82] was also required.

2. Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with mail programs that assume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really *are* used and are difficult to support, but must be supported.

3. Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.

1. This section makes no distinction between *delivermail* and *sendmail.*

4. Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ethernets). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use.

5. Configuration information should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to ''fiddle'' with anything that they will be recompiling anyway.

6. *Sendmail* must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.

7. Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an ''I am on vacation'' message).

8. Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in Figure 1-1.

```
+----------+   +----------+   +----------+
| sender1  |   | sender2  |   | sender3  |
+----------+   +----------+   +----------+
     |              |              |
     +----------+   +   +----------+
                |   |   |
                v   v   v
            +-------------+
            |  sendmail   |
            +-------------+
                |   |   |
     +----------+   +   +----------+
     |              |              |
     v              v              v
+----------+   +----------+   +----------+
| mailer1  |   | mailer2  |   | mailer3  |
+----------+   +----------+   +----------+
```

Figure 2-1. Sendmail System Structure

The user interacts with a mail generating and sending program. When the mail is created, the generator calls *sendmail*, which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, *sendmail* may be used as an internet mail gateway.

3. Usage

3.1 Address Formats

Arguments may be flags or addresses. Flags set various processing options. Following flag arguments, address arguments may be given. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

1. Anything in parentheses is thrown away (as a comment).

2. Anything in angle brackets (‘‘<>’’) is preferred over anything else. This rule implements the ARPANET standard that addresses of the form

   ```
   user name <machine-address>
   ```

   will send to the electronic ‘‘machine-address’’ rather than the human ‘‘user name.’’

3. Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently – for example, *user* and ‘‘*user*’’ are equivalent, but ‘‘\user’’ is different from either of them. This might be used to avoid normal aliasing or duplicate suppression algorithms.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing.[1]

Although old style addresses are still accepted in most cases, the preferred address format is based on ARPANET-style domain-based addresses [Su82a]. These addresses are based on a hierarchical, logical decomposition of the address space. The addresses are hierarchical in a sense similar to the U.S. postal addresses: the messages may first be routed to the correct state, with no initial consideration of the city or other addressing details. The addresses are logical in that each step in the hierarchy corresponds to a set of ''naming authorities'' rather than a physical network.

For example, the address:

```
eric@HostA.BigSite.ARPA
```

would first look up the domain BigSite in the namespace administrated by ARPA. A query could then be sent to BigSite for interpretation of HostA. Eventually the mail would arrive at HostA, which would then do final delivery to user ''eric''.

3.2 Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley *msgs* program).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar (''|)'' the rest of the address is processed as a shell command. If the user name begins with a slash mark (''\'') the name is used as a file name, instead of a login name.

1. Disclaimer: Some special processing is done after rewriting local names; see below.

3.3 Aliasing, Forwarding, Inclusion

Sendmail reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs *sendmail* to read a file for a list of addresses, and is normally used in conjunction with aliasing.

3.3.1 Aliasing

Aliasing maps local addresses to address lists using a system-wide file. This file is hashed to speed access. Only addresses that parse as local are allowed as aliases; this guarantees a unique key (since there are no nicknames for the local host).

3.3.2 Forwarding

After aliasing, if an recipient address specifies a local user *sendmail* searches for a ''.forward'' file in the recipient's home directory. If it exists, the message is *not* sent to that user, but rather to the list of addresses in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
"| /usr/local/newmail myname"
```

will use a different incoming mailer.

3.3.3 Inclusion

Inclusion is specified in RFC 733 [Crocker77] syntax:

```
:Include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is *not* to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

```
project: :include:/usr/project/userlist
```

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

3.4 Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line. The body is an uninterpreted sequence of text lines.

The header is formated as a series of lines of the form

```
field-name: field-value
```

Field-value can be split across lines by starting the following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

4. The UUCP Problem

Of particular interest is the UUCP network. The explicit routing used in the UUCP environment causes a number of serious problems. First, giving out an address is impossible without knowing the address of your potential correspondent. This is typically handled by specifying the address relative to some ''well-known'' host (e.g., ucbvax or decvax). Second, it is often difficult to compute the set of addresses to reply to without some knowledge of the topology of the network. Although it may be easy for a human being to do this under many circumstances, a program does not have equally sophisticated heuristics built in. Third, certain addresses will become painfully and unnecessarily long, as when a message is routed through many hosts in the USENET. And finally, certain ''mixed domain'' addresses are impossible to parse unambiguously – e.g.,

```
decvax!ucbvax!lbl-h!user@LBL-CSAM
```

might have many possible resolutions, depending on whether the message was first routed to decvax or to LBL-CSAM.

To solve this problem, the UUCP syntax would have to be changed to use addresses rather than routes. For example, the address *decvax!ucbvax!eric* might be expressed as *eric@ucbvax.UUCP* (with the hop through decvax implied). This address would itself be a domain-based address; for example, an address might be of the form:

```
mark@d.cbosg.btl.UUCP
```

Hosts outside of Bell Telephone Laboratories would then only need to know how to get to a designated BTL relay, and the BTL topology would only be maintained inside Bell.

There are three major problems associated with turning UUCP addresses into something reasonable: defining the namespace, creating and propagating the necessary software, and building and maintaining the database.

4.1 Defining the Namespace

Putting all UUCP hosts into a flat namespace (e.g., "*...@host.UUCP*") is not practical for a number of reasons. First, with over 1600 sites already, and (with the increasing availability of inexpensive microcomputers and autodialers) several thousand more coming within a few years, the database update problem is simply intractable if the namespace is flat. Second, there are almost certainly name conflicts today. Third, as the number of sites grow the names become ever less mnemonic.

It seems inevitable that there be some sort of naming authority for the set of top level names in the UUCP domain, as unpleasant a possibility as that may seem. It will simply not be possible to have one host resolving all names. It may however be possible to handle this in a fashion similar to that of assigning names of newsgroups in USENET. However, it will be essential to encourage everyone to become subdomains of an existing domain whenever possible – even though this will certainly bruise some egos. For example, if a new host named *blid* were to be added to the UUCP network, it would probably actually be addressed as *d.bli.UUCP* (i.e., as host *d* in the pseudo-domain *bli* rather than as host *blid* in the UUCP domain).

4.2 Creating and Propagating the Software

The software required to implement a consistent namespace is relatively trivial. Two modules are needed, one to handle incoming mail and one to handle outgoing mail.

The incoming module must be prepared to handle either old or new style addresses. New-style addresses can be passed through unchanged. Old style addresses must be turned into new style addresses where possible.

The outgoing module is slightly trickier. It must do a database lookup on the recipient addresses (passed on the command line) to determine what hosts to send the message to. If those hosts do not accept new-style addresses, it must transform all addresses in the header of the message into old style using the database lookup.

Both of these modules are straightforward except for the issue of modifying the header. It seems prudent to choose one format for the message headers. For a number of reasons, Berkeley has elected to use the ARPANET protocols for message formats. However, this protocol is somewhat difficult to parse.

Propagation is somewhat more difficult. There are a large number of hosts connected to UUCP that will want to run completely standard systems (for very good reasons). The strategy is not to convert the entire network – only enough of it it alleviate the problem.

4.3 Building and Maintaining the Database

This is by far the most difficult problem. A prototype for this database already exists, but it is maintained by hand and does not pretend to be complete.

This problem will be reduced considerably if people choose to group their hosts into subdomains. This would require a global update only when a new top level domain joined the network. A message to a host in a subdomain could simply be routed to a known domain gateway for further processing. For example, the address *eric@a.bli.UUCP* might be routed to the *bli* gateway for redistribution; new hosts could be added within BLI without notifying the rest of the world. Of course, other hosts *could* be notified as an efficiency measure.

There may be more than one domain gateway. A domain such as BTL, for instance, might have a dozen gateways to the outside world; a non-BTL site could choose the closest gateway. The only restriction would be that all gateways maintain a consistent view of the domain they represent.

4.4 Logical Structure

Logically, domains are organized into a tree. There need not be a host actually associated with each level in the tree – for example, there will be no host associated with the name *UUCP*. Similarly, an organization might group names together for administrative reasons; for example, the name

```
CAD.research.BigCorp.UUCP
```

might not actually have a host representing *research.*

However, it may frequently be convenient to have a host or hosts that "represent" a domain. For example, if a single host exists that represents Berkeley, then mail from outside Berkeley can forward mail to that host for further resolution without knowing Berkeley's (rather volatile) topology. This is not unlike the operation of the telephone network.

This may also be useful inside certain large domains. For example, at Berkeley it may be presumed that most hosts know about other hosts inside the Berkeley domain. But if they process an address that is unknown, they can pass it "upstairs" for further examination. Thus as new hosts are added only one host (the domain master) *must* be updated immediately; other hosts can be updated as convenient.

Ideally this name resolution process would be performed by a name server (e.g., [Su82b]) to avoid unnecessary copying of the message. However, in a batch network such as UUCP this could result in unnecessary delays.

5. Comparison with Delivermail

Sendmail is an outgrowth of *delivermail.* The primary differences are:

1. Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also allows easy debugging of new mailers.

2. Address parsing is more flexible. For example, *delivermail* only supported one gateway to any network, whereas *sendmail* can be sensitive to host names and reroute to different gateways.

3. Forwarding and :include: features eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).

4. *Sendmail* supports message batching across networks when a message is being sent to multiple recipients.

5. A mail queue is provided in *sendmail.* Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected it may be reliably redelivered even if the system crashes during the initial delivery.

6. *Sendmail* uses the networking support provided by 4.2BSD to provide a direct interface networks such as the ARPANET and/or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

REFERENCES

[Crocker77] Crocker, D. H., Vittal, J. J., Pogran, K. T., and Henderson, D. A. Jr.,

Standard for the Format of ARPA Network Text Messages. RFC 733, NIC 41952. In [Feinler78]. November 1977. [Crocker82] Crocker, D. H.,

Standard for the Format of Arpa Internet Text Messages. RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982. [Feinler78] Feinler, E., and Postel, J. (eds.),

ARPANET Protocol Handbook. NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978. [Nowitz78] Nowitz, D. A., and Lesk, M. E.,

A Dial-Up Network of UNIX Systems. Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. August, 1978. [Schmidt79] Schmidt, E.,

An Introduction to the Berkeley Network. University of California, Berkeley California. 1979. [Shoens79] Shoens, K.,

Mail Reference Manual. University of California, Berkeley. In UNIX Programmer's Manual, Seventh Edition, Volume 2C. December 1979. [Solomon81] Solomon, M., Landweber, L., and Neuhengen, D.,

The Design of the CSNET Name Server. CS-DN-2. University of Wisconsin, Madison. October 1981. [Su82a] Su, Zaw-Sing, and Postel, Jon,

The Domain Naming Convention for Internet User Applications. RFC819. Network Information Center, SRI International, Menlo Park, California. August 1982. [Su82b] Su, Zaw-Sing,

A Distributed System for Internet Name Service. RFC830. Network Information Center, SRI International, Menlo Park, California. October 1982.

Sendmail Guide

Sendmail Installation and Operation Guide SG-1

1. Basic Installation SG-3
1.1 Off-The-Shelf Configurations SG-3
1.2 Installation Using the Makefile SG-4
1.3 Installation by Hand SG-4
1.3.1 *lib/libsys.a* SG-4
1.3.2 */usr/lib/sendmail* SG-5
1.3.3 */usr/lib/sendmail.cf* SG-5
1.3.4 */usr/ucb/newaliases* SG-5
1.3.5 */usr/spool/mqueue* SG-6
1.3.6 */usr/lib/aliases* SG-6
1.3.7 */usr/lib/sendmail.fc* SG-6
1.3.8 */etc/rc* SG-7
1.3.9 */usr/lib/sendmail.hf* SG-7
1.3.10 */usr/lib/sendmail.st* SG-7
1.3.11 */etc/syslog* SG-8
1.3.12 */usr/ucb/newaliases* SG-8
1.3.13 */usr/ucb/mailq* SG-8

2. Normal Operations SG-9
2.1 Quick Configuration Startup SG-9
2.2 The System Log SG-9
2.2.1 Format SG-9
2.2.2 Levels SG-10
2.3 The Mail Queue SG-10
2.3.1 Printing the queue SG-10
2.3.2 Format of queue files SG-10
2.3.3 Forcing the queue SG-12
2.4 The Alias Database SG-13
2.4.1 Rebuilding the alias database SG-14
2.4.2 Potential problems SG-14

2.4.3 List owners SG-15
2.5 Per-User Forwarding (*.forward Files*) SG-15
2.6 Special Header Lines SG-16
2.6.1 Return-Receipt-To: SG-16
2.6.2 Errors-To: SG-16
2.6.3 Apparently-To: SG-16

3. Arguments SG-17
3.1 Queue Interval SG-17
3.2 Daemon Mode SG-17
3.3 Forcing the Queue SG-18
3.4 Debugging SG-18
3.5 Trying a Different Configuration File SG-19
3.6 Changing the Values of Options SG-19

4. Tuning SG-21
4.1 Timeouts SG-21
4.1.1 Queue interval SG-21
4.1.2 Read timeouts SG-22
4.1.3 Message timeouts SG-22
4.2 Delivery Mode SG-22
4.3 Log Level SG-23
4.4 Load Limiting SG-23
4.5 File Modes SG-24
4.5.1 To *suid* or not to *suid*? SG-24
4.5.2 Temporary file modes SG-24
4.5.3 Should my alias database be writable? SG-24

5. The Whole Scoop on the Configuration File SG-27
5.1 The Syntax SG-27
5.1.1 R and S — rewriting rules SG-27
5.1.2 D — define macro SG-28
5.1.3 C and F — define classes SG-28
5.1.4 M — define mailer SG-29
5.1.5 H — define header SG-30

5.1.6 O — set option SG-30
5.1.7 T — define trusted users SG-30
5.1.8 P — precedence definitions SG-31
5.2 The Semantics SG-31
5.2.1 Special macros, conditionals SG-31
5.2.2 Special classes SG-34
5.2.3 The left hand side SG-35
5.2.4 The right hand side SG-35
5.2.5 Semantics of rewriting rule sets SG-37
5.2.6 Mailer flags etc. SG-38
5.2.7 The “error” mailer SG-38
5.3 Building a Configuration File From Scratch SG-38
5.3.1 What you are trying to do SG-38
5.3.2 Philosophy SG-39
5.3.3 Relevant issues SG-41
5.3.4 How to proceed SG-42
5.3.5 Testing the rewriting rules — the -bt flag SG-42
5.3.6 Building mailer descriptions SG-43

A: Command Line Flags SG-47

B: Configuration Options SG-49

C: Mailer Flags SG-53

D: Other Configuration SG-57

E: Summary of Support Files SG-63

Sendmail Installation and Operation Guide

Eric Allman
Britton-Lee, Inc.

Version 5.1

Abstract

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.

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.

1.1 Off-The-Shelf Configurations

The configuration files are all in the subdirectory *cf* of the sendmail directory. The ones used at Berkeley are in *m4* (1) format; files with names ending *.m4* are *m4* include files, while files with names ending *.mc* are the master files. Files with names ending *.cf* are the *m4* processed versions of the corresponding *.mc* file.

Two off the shelf configuration files are supplied to handle the basic cases: *cf/arpaproto.cf* for Arpanet (TCP) sites and *cf/uucpproto.cf* for UUCP sites. These are *not* in *m4* format. The file you need should be copied to a file with the same name as your system, e.g.,

```
cp  uucpproto.cf  ucsfcgl.cf
```

This file is now ready for installation as */usr/lib/sendmail.cf.*

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 4.2bsd system. It may have to be slightly tailored for use on other systems.

Before using this makefile, you should already 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
```

in the root directory of the *sendmail* distribution. This will make all binaries and install them in the standard places. The second *make* command 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 *lib/libsys.a*

The library in *lib/libsys.a* contains some routines that should in some sense be part of the system library. These are the system logging routines and the new directory access routines (if required). If you are not running the new

4.2bsd directory code and do not have the compatibility routines installed in your system library, you should execute the commands:

```
cd  lib
make  ndir
```

This will compile and install the 4.2 compatibility routines in the library. You should then type:

```
cd  lib      # if required
make
```

This will recompile and fill the library.

1.3.2 */usr/lib/sendmail*

The binary for sendmail is located in */usr/lib*. There is a version available in the source directory that is probably inadequate for your system. You should plan on recompiling and installing the entire system:

```
cd  src
rm  -f  *.o
make
cp  sendmail  /usr/lib
```

1.3.3 */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
```

1.3.4 */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
```

1.3.5 */usr/spool/mqueue*

The directory */usr/spool/mqueue* should be created to hold the mail queue. This directory should be mode 777 unless *sendmail* is run setuid, when *mqueue* should be owned by the sendmail owner and mode 755.

1.3.6 */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 666 if you are running a reasonably relaxed system:

```
cp  /dev/null  /usr/lib/aliases.dir
cp  /dev/null  /usr/lib/aliases.pag
chmod  666  /usr/lib/aliases.*
newaliases
```

1.3.7 */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
```

1.3.8 */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.

1.3.9 */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
```

1.3.10 */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*.

1.3.11 */etc/syslog*

You may want to run the *syslog* program (to collect log information about sendmail). This program normally resides in */etc/syslog*, with support files */etc/syslog.conf* and */etc/syslog.pid*. The program is located in the *aux* subdirectory of the *sendmail* distribution. The file */etc/syslog.conf* describes the file(s) that sendmail will log in. For a complete description of syslog, see the manual page for *syslog* (3) (located in *sendmail/doc* on the distribution).

1.3.12 */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*.

1.3.13 */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 *syslog* (3) 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 ethernet), the word *sendmail:*, and a message.

2.2.2 Levels

If you have *syslog* (3) 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 4.3.

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.
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 increases 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.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
                    id A13557; 23-Oct-82 15:49:32-PDT (Sat)
Hphone: (415) 548-3211
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 777 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
```

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* (3X) 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 maintanence 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 *file with contents:*

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

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

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.

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. if the mailer has 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.

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).

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 five minutes and one half 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 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.

4.3 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 defered (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.

4.4 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) until the load drops.

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

4.5 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.

4.5.1 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.

4.5.2 Temporary file modes

The mode of all temporary files that *sendmail* creates is determined by the *F* option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it will not be necessary to run *sendmail* as root at all (even when running the queue).

4.5.3 Should my alias database be writable?

At Berkeley we have the alias database *(/usr/lib/aliases*)* mode 666. There are some dangers inherent in this approach: any user can add him-/her-self to any list, or can *steal* any other user's mail. However, we have found users to be basically trustworthy, and the cost of having a read-only database greater than the expense of finding and eradicating the rare nasty person.

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. 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 *word1 word2...*
F *file* [*format*]

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*; the *format* is a *scanf*(3) pattern that should produce a single string.

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:

```
Oo 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:

```
Tuser1  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.

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 *r* were in the **$o** macro, then the input *address* would be scanned as three tokens: *add, r,* and *ess.* Finally, the **$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 Arpanet format
b	The current date in Arpanet 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 Arpanet 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.

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.

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 delete local hostnames.

5.2.3 The left hand side

The left hand side of rewriting rules contains a [illegible]rn. 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 are 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 $] is looked up in the */etc/hosts* file and replaced by the canonical name. For example, ''$[csam$]'' would become ''lbl-csam.arpa''.

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 preceeded 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 preceed 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 the figure below.

```
                  +---+
               -->| 0 |-->resolved address
              /   +---+
             /          +---+  +---+
            /       --->| 1 |->| S |-
      +---+/ +---+ /    +---+  +---+ \   +---+
addr->| 3 |->| D |-                   -->| 4 |->msg
      +---+  +---+ \    +---+  +---+ /   +---+
                    --->| 2 |->| R |-
                        +---+  +---+
```

Figure 5-1. Rewriting set semantics

```
D - sender domain addition
S - mailer-specific sender rewriting
R - mailer-specific recipient rewriting
```

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.

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.

Large site, many hosts – minimum information

Berkeley is an example of a large site, i.e., more than two or three hosts. 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.* Monet has a list of known ethernet hosts; if it receives mail for any of them, it can do direct delivery. 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 ethernet host is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated as convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, our TCP connection is 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 TCP host is transfered via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to the Arpanet. 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 TCP mail directly to ucbarpa if the load got too high; if monet failed to note a host as a TCP host 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, as if ucbarpa thought that ucbvax had the TCP 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.

Small site – complete information

A small site (two or three hosts) 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.

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 is *a.cc.berkeley.arpa*; reading from right to left, *arpa* is a top level domain (related to, but not limited to, the physical Arpanet), *berkeley* is both an Arpanet host and a logical domain which is actually interpreted by a host called ucbvax (which is actually just the *major* host for this domain), *cc* represents the Computer Center, (in this case a strictly logical entity), and *a* is a host in the Computer Center; this particular host happens to be connected via berknet, but other hosts might be connected via one of two ethernets or some other network.

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. If this mailer is local [1] (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

1. *Sendmail* must be running setuid to root for this to work.

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. [2]

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
```

2. The *c* configuration option must be given for this to be effective.

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 ]
```

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*n*	The temporary file mode, in octal. 644 and 600 are good choices.
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	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 timezone 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.
z	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.

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 three places:

md/config.m4	These contain operating-system dependent descriptions. They are interpolated into the Makefiles in the *src* and *aux* directories. This includes information about what version of UNIX you are running, what libraries you have to include, etc.
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 *md/config.m4*

The following compilation flags may be defined in the *m4CONFIG* macro in *md/config.m4* to define the environment in which you are operating.

V6	If set, this will compile a version 6 system, with 8-bit user id's, single character tty id's, etc.
VMUNIX	If set, you will be assumed to have a Berkeley 4BSD or 4.1BSD, including the *vfork*(2) system call, special types defined in <*sys/types.h*> (e.g, *u_char*), etc.

If none of these flags are set, a version 7 system is assumed.

You will also have to specify what libraries to link with *sendmail* in the *m4LIBS* macro. Most notably, you will have to include **–ljobs** if you are running a 4.1BSD system.

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 [256]	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 [128]	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 [30]	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*).

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.
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 4.2BSD if the NVMUNIX flag is specified; otherwise, 4.1a BSD code is used. Beware however that there are bugs in the 4.1a code that make it impossible for **sendmail** to work correctly under heavy load.
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.

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 (transient) 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, whereas *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 supress 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.

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.
/etc/syslog	The program to do logging.
/etc/syslog.conf	The configuration file for syslog.
/etc/syslog.pid	Contains the process id of the currently running syslog.
/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.

Sendmail Router

Sendmail—An Internetwork Mail Router SR-1

1. Design Goals SR-3

2. Overview SR-7
2.1 System Organization SR-7
2.2 Interfaces to the Outside World SR-7
2.2.1 Argument vector/exit status SR-8
2.2.2 SMTP over pipes SR-8
2.2.3 SMTP over an IPC connection SR-8
2.3 Operational Description SR-8
2.3.1 Argument processing and address parsing SR-9
2.3.2 Message collection SR-9
2.3.3 Message delivery SR-9
2.3.4 Queueing for retransmission SR-10
2.3.5 Return to sender SR-10
2.4 Message Header Editing SR-10
2.5 Configuration File SR-11

3. Usage and Implementation SR-13
3.1 Arguments SR-13
3.2 Mail to Files and Programs SR-14
3.3 Aliasing, Forwarding, Inclusion SR-14
3.3.1 Aliasing SR-15
3.3.2 Forwarding SR-15
3.3.3 Inclusion SR-15
3.4 Message Collection SR-16
3.5 Message Delivery SR-16
3.6 Queued Messages SR-17
3.7 Configuration SR-17

3.7.1 Macros SR-18
3.7.2 Header declarations SR-18
3.7.3 Mailer declarations SR-18
3.7.4 Address rewriting rules SR-18
3.7.5 Option setting SR-19

4. Comparison with Other Mailers SR-21
4.1 Delivermail SR-21
4.2 MMDF SR-22
4.3 Message Processing Module SR-22

5. Evaluations and Future Plans SR-25
5.1 Acknowledgements SR-27

Sendmail—An Internetwork Mail Router

Eric Allman[†]

Britton-Lee, Inc.
1919 Addison Street, Suite 105.
Berkeley, California 94704

Abstract

Routing mail through a heterogenous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an *ad hoc* basis. However, this approach has become unmanageable as internets grow.

Sendmail acts a unified "post office" to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both domain-based addressing and old-style *ad hoc* addresses. The production system is powerful enough to rewrite addresses in the message header to conform to the standards of a number of common target networks, including old (NCP/RFC733) Arpanet, new (TCP/RFC822) Arpanet, UUCP, and Phonenet. Sendmail also implements an SMTP server, message queueing, and aliasing.

Sendmail implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

---

† A considerable part of this work was done while under the employ of the INGRES Project at the University of California at Berkeley.

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characterstics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require point-to-point routing, which simplifies the database update problem since only adjacent hosts must be entered into the system tables, while others use end-to-end addressing. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

Internet standards seek to eliminate these problems. Initially, these proposed expanding the address pairs to address triples, consisting of {network, host, resource} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was quickly expanded to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root. The domain technique separates the issue of physical versus logical addressing. For example, an address of the form *eric@a.cc.berkeley.arpa* describes only the logical organization of the address space.

Sendmail is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, *sendmail* is designed to assist a graceful transition to consistent internetwork addressing schemes.

Section 1 discusses the design goals for *sendmail*. Section 2 gives an overview of the basic functions of the system. In section 3, details of usage are discussed. Section 4 compares *sendmail* to other internet mail routers, and an evaluation of *sendmail* is given in section 5, including future plans.

1. Design Goals

Design goals for *sendmail* include:

1. Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail [UNIX83], Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78a, Nowitz78b]. ARPANET mail [Crocker77a, Postel77] was also required.

2. Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with UNIX mail programs that assume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really *are* used and are difficult to support, but must be supported.

3. Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.

4. Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ether nets [Metcalfe76]). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use. For example, the ARPANET is bringing up the TCP protocol to replace the old NCP protocol. No host at Berkeley runs both TCP and NCP, so it is necessary to look at the ARPANET host name to determine whether to route mail to an NCP gateway or a TCP gateway.

5. Configuration should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to ''fiddle'' with anything that they will be recompiling anyway.

6. *Sendmail* must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.

7. Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an "I am on vacation" message).

8. Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in figure 1-1.

```
+----------+  +----------+  +----------+
| sender1  |  | sender2  |  | sender3  |
+----------+  +----------+  +----------+
     |             |             |
     +----------+  +  +----------+
                |  |  |
                v  v  v
            +-------------+
            |   sendmail  |
            +-------------+
                |  |  |
     +----------+  +  +----------+
     |             |             |
     v             v             v
+----------+  +----------+  +----------+
| mailer1  |  | mailer2  |  | mailer3  |
+----------+  +----------+  +----------+
```

Figure 1-1. Sendmail System Structure

The user interacts with a mail generating and sending program. When the mail is created, the generator calls *sendmail,* which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, may be used as an internet mail gateway.

2. Overview

2.1 System Organization

Sendmail neither interfaces with the user nor does actual mail delivery. Rather, it collects a message generated by a user interface program (UIP) such as Berkeley *Mail*, MS [Crocker77b], or MH [Borden79], edits the message as required by the destination network, and calls appropriate mailers to do mail delivery or queueing for network transmission. [1] This discipline allows the insertion of new mailers at minimum cost. In this sense *sendmail* resembles the Message Processing Module (MPM) of [Postel79b].

2.2 Interfaces to the Outside World

There are three ways *sendmail* can communicate with the outside world, both in receiving and in sending mail. These are using the conventional UNIX argument vector/return status, speaking SMTP over a pair of UNIX pipes, and speaking SMTP over an interprocess(or) channel.

1. except when mailing to a file, when *sendmail* does the delivery directly.

2.2.1 Argument vector/exit status

This technique is the standard UNIX method for communicating with the process. A list of recipients is sent in the argument vector, and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

2.2.2 SMTP over pipes

The SMTP protocol [Postel82] can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the processes standard input. Anything appearing on the standard output must be a reply code in a special format.

2.2.3 SMTP over an IPC connection

This technique is similar to the previous technique, except that it uses a 4.2bsd IPC channel [UNIX83]. This method is exceptionally flexible in that the mailer need not reside on the same machine. It is normally used to connect to a sendmail process on another machine.

2.3 Operational Description

When a sender wants to send a message, it issues a request to *sendmail* using one of the three methods described above. *Sendmail* operates in two distinct phases. In the first phase, it collects and stores the message. In the second phase, message delivery occurs. If there were errors during processing during the second phase, *sendmail* creates and returns a new message describing the error and/or returns an status code telling what went wrong.

2.3.1 Argument processing and address parsing

If *sendmail* is called using one of the two subprocess techniques, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP RCPT command, and a list of recipients is created. Aliases are expanded at this step, including mailing lists. As much validation as possible of the addresses is done at this step: syntax is checked, and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

Sendmail appends each address to the recipient list after parsing. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages deliverd to the same recipient, as might occur if a person is in two groups.

2.3.2 Message collection

Sendmail then collects the message. The message should have a header at the beginning. No formatting requirements are imposed on the message except that they must be lines of text (i.e., binary data is not allowed). The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses were valid. The message will be returned with an error.

2.3.3 Message delivery

For each unique mailer and host in the recipient list, *sendmail* calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that only accept one recipient at a time are handled properly.

The message is sent to the mailer using one of the same three interfaces used to submit a message to sendmail. Each copy of the message is prepended by a customized header. The mailer status code is caught and checked, and a suitable error message given as appropriate. The exit code must conform to a system standard or a generic message (''Service unavailable'') is given.

2.3.4 Queueing for retransmission

If the mailer returned an status that indicated that it might be able to handle the mail later, *sendmail* will queue the mail and try again later.

2.3.5 Return to sender

If errors occur during processing, *sendmail* returns the message to the sender for retransmission. The letter can be mailed back or written in the file ''dead.letter'' in the sender's home directory. [2]

2.4 Message Header Editing

Certain editing of the message header occurs automatically. Header lines can be inserted under control of the configuration file. Some lines can be merged; for example, a ''From:'' line and a ''Full-name:'' line can be merged under certain circumstances.

2. Obviously, if the site giving the error is not the originating site, the only reasonable option is to mail back to the sender. Also, there are many more error disposition options, but they only effect the error message – the ''return to sender'' function is always handled in one of these two ways.

2.5 Configuration File

Almost all configuration information is read at runtime from an ASCII file, encoding macro definitions (defining the value of macros used internally), header declarations (telling sendmail the format of header lines that it will process specially, i.e., lines that it will add or reformat), mailer definitions (giving information such as the location and characteristics of each mailer), and address rewriting rules (a limited production system to rewrite addresses which is used to parse and rewrite the addresses).

To improve performance when reading the configuration file, a memory image can be provided. This provides a ''compiled'' form of the configuration file.

3. Usage and Implementation

3.1 Arguments

Arguments may be flags and addresses. Flags set various processing options. Following flag arguments, address arguments may be given, unless we are running in SMTP mode. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

1. Anything in parentheses is thrown away (as a comment).

2. Anything in angle brackets ("<>") is preferred over anything else. This rule implements the ARPANET standard that addresses of the form

   ```
   user name <machine-address>
   ```

   will send to the electronic ''machine-address'' rather than the human ''user name''.

3. Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently – for example, *user* and ''*user*'' are equivalent, but \user is different from either of them.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing. [1]

3.2 Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley *msgs* program, or the MARS system [Sattley78]).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar (``|'' ) the rest of the address is processed as a shell command. If the user name begins with a slash mark (``/'') the name is used as a file name, instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if *sendmail* is running as root.

3.3 Aliasing, Forwarding, Inclusion

Sendmail reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs *sendmail* to read a file for a list of addresses, and is normally used in conjunction with aliasing.

1. Disclaimer: Some special processing is done after rewriting local names; see below.

3.3.1 Aliasing

Aliasing maps names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases; this guarantees a unique key (since there are no nicknames for the local host).

3.3.2 Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a *.forward* file in their home directory. If it exists, the message is *not* sent to that user, but rather to the list of users in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
"| /usr/local/newmail myname"
```

will use a different incoming mailer.

3.3.3 Inclusion

Inclusion is specified in RFC 733 [Crocker77a] syntax:

```
:Include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is *not* to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

```
project: :include:/usr/project/userlist
```

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

3.4 Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line.

The header is formatted as a series of lines of the form

```
field-name: field-value
```

Field-value can be split across lines by starting the following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

The body is a series of text lines. It is completely uninterpreted and untouched, except that lines beginning with a dot have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver.

3.5 Message Delivery

The send queue is ordered by receiving host before transmission to implement message batching. Each address is marked as it is sent so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list. The interface to the mailer is performed using one of the techniques described in section 2.2.

After a connection is established, *sendmail* makes the per-mailer changes to the header and sends the result to the mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

3.6 Queued Messages

If the mailer returns a ''temporary failure'' exit status, the message is queued. A control file is used to describe the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other salient parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

3.7 Configuration

Configuration is controlled primarily by a configuration file read at startup. *Sendmail* should not need to be recomplied except

1. To change operating systems (V6, V7/32V, 4BSD).
2. To remove or insert the DBM (UNIX database) library.
3. To change ARPANET reply codes.
4. To add headers fields requiring special processing.
5. Adding mailers or changing parsing (i.e., rewriting) or routing information does not require recompilation.

If the mail is being sent by a local user, and the file *.mailcf* exists in the sender's home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options.

3.7.1 Macros

Macros can be used in three ways. Certain macros transmit unstructured textual information into the mail system, such as the name *sendmail* will use to identify itself in error messages. Other macros transmit information from *sendmail* to the configuration file for use in creating other fields (such as argument vectors to mailers); e.g., the name of the sender, and the host and user of the recipient. Other macros are unused internally, and can be used as shorthand in the configuration file.

3.7.2 Header declarations

Header declarations inform *sendmail* of the format of known header lines. Knowledge of a few header lines is built into *sendmail*, such as the "From:" and "Date:" lines.

Most configured headers will be automatically inserted in the outgoing message if they don't exist in the incoming message. Certain headers are suppressed by some mailers.

3.7.3 Mailer declarations

Mailer declarations tell *sendmail* of the various mailers available to it. The definition specifies the internal name of the mailer, the pathname of the program to call, some flags associated with the mailer, and an argument vector to be used on the call; this vector is macro-expanded before use.

3.7.4 Address rewriting rules

The heart of address parsing in *sendmail* is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address. The address is rewritten textually until it is either rewritten into a special canonical form (i.e., a (mailer, host, user) 3-tuple, such as {arpanet, usc-isif, postel} representing the address *postel@usc-isif*) or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

```
ucsfcgl!tef
```

might be mapped into:

```
tef@ucsfcgl.UUCP
```

to conform to the domain syntax. Translations can also be done in the other direction.

3.7.5 Option setting

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes, etc.

4. Comparison with Other Mailers

4.1 Delivermail

Sendmail is an outgrowth of *delivermail.* The primary differences are:

1. Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also allows easy debugging of new mailers.
2. Address parsing is more flexible. For example, *delivermail* only supported one gateway to any network, whereas *sendmail* can be sensitive to host names and reroute to different gateways.
3. Forwarding and :include: features eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).
4. *Sendmail* supports message batching across networks when a message is being sent to multiple recipients.
5. A mail queue is provided in *sendmail.* Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected it may be reliably redelivered even if the system crashes during the initial delivery.
6. *Sendmail* uses the networking support provided by 4.2BSD to provide a direct interface networks such as the ARPANET and/or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

4.2 MMDF

MMDF [Crocker79] spans a wider problem set than *sendmail.* For example, the domain of MMDF includes a ''phone network'' mailer, whereas *sendmail* calls on preexisting mailers in most cases.

MMDF and *sendmail* both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which *sendmail* does not support.

The configuration for MMDF is compiled into the code. [1]

Since MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel [2] into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and the channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although *sendmail* has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

4.3 Message Processing Module

The Message Processing Module (MPM) discussed by Postel [Postel79b] matches *sendmail* closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF, thus allowing simpler handling of errors by the mailer than is possible in

1. Dynamic configuration tables are currently being considered for MMDF; allowing the installer to select either compiled or dynamic tables.
2. The MMDF equivalent of a *sendmail* '' mailer''.

sendmail. When a message queued by *sendmail* is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with type-length-value tuples. [3] Such a convention requires a much higher degree of cooperation between mailers than is required by *sendmail.* MPM also assumes a universally agreed upon internet name space (with each address in the form of a net-host-user tuple), which *sendmail* does not.

3. This is similar to the NBS standard.

5. Evaluations and Future Plans

Sendmail is designed to work in a nonhomogeneous environment. Every attempt is made to avoid imposing unnecessary constraints on the underlying mailers. This goal has driven much of the design. One of the major problems has been the lack of a uniform address space, as postulated in [Postel79a] and [Postel79b].

A nonuniform address space implies that a path will be specified in all addresses, either explicitly (as part of the address) or implicitly (as with implied forwarding to gateways). This restriction has the unpleasant effect of making replying to messages exceedingly difficult, since there is no one ''address'' for any person, but only a way to get there from wherever you are.

Interfacing to mail programs that were not initially intended to be applied in an internet environment has been amazingly successful, and has reduced the job to a manageable task.

Sendmail has knowledge of a few difficult environments built in. It generates ARPANET FTP/SMTP compatible error messages (prepended with three-digit numbers [Neigus73, Postel74, Postel82]) as necessary, optionally generates UNIX-style ''From'' lines on the front of messages for some mailers, and knows how to parse the same lines on input. Also, error handling has an option customized for BerkNet.

The decision to avoid doing any type of delivery where possible (even, or perhaps especially, local delivery) has turned out to be a good idea. Even with local delivery, there are issues of the location of the mailbox, the format of the mailbox, the locking protocol used, etc., that are best decided by other programs. One surprisingly major annoyance in many internet mailers is that the location and format of local mail is built in. The feeling seems to be that local mail is so common that it should be efficient. This feeling is not born out by our experience; on the contrary, the location and format of mailboxes seems to vary widely from system to system.

The ability to automatically generate a response to incoming mail (by forwarding mail to a program) seems useful (''I am on vacation until late August....'') but can create problems such as forwarding loops (two people on vacation whose programs send notes back and forth, for instance) if these programs are not well written. A program could be written to do standard tasks correctly, but this would solve the general case.

It might be desirable to implement some form of load limiting. I am unaware of any mail system that addresses this problem, nor am I aware of any reasonable solution at this time.

The configuration file is currently practically inscrutable; considerable convenience could be realized with a higher-level format.

It seems clear that common protocols will be changing soon to accommodate changing requirements and environments. These changes will include modifications to the message header (e.g., [NBS80]) or to the body of the message itself (such as for multimedia messages [Postel80]). Experience indicates that these changes should be relatively trivial to integrate into the existing system.

In tightly coupled environments, it would be nice to have a name server such as Grapvine [Birrell82] integrated into the mail system. This would allow a site such as *Berkeley* to appear as a single host, rather than as a collection of hosts, and would allow people to move transparently among machines without having to change their addresses. Such a facility would require an automatically updated database and some method of resolving conflicts. Ideally this would be effective even without all hosts being under a single management. However, it is not clear whether this feature should be integrated into the aliasing facility or should be considered a ''value added'' feature outside *sendmail* itself.

As a more interesting case, the CSNET name server [Solomon81] provides an facility that goes beyond a single tightly-coupled environment. Such a facility would normally exist outside of *sendmail* however.

5.1 Acknowledgements

Thanks are due to Kurt Shoens for his continual cheerful assistance and good advice, Bill Joy for pointing me in the correct direction (over and over), and Mark Horton for more advice, prodding, and many of the good ideas. Kurt and Eric Schmidt are to be credited for using *delivermail* as a server for their programs (*Mail* and BerkNet respectively) before any sane person should have, and making the necessary modifications promptly and happily. Eric gave me considerable advice about the perils of network software which saved me an unknown amount of work and grief. Mark did the original implementation of the DBM version of aliasing, installed the VFORK code, wrote the current version of *rmail*, and was the person who really convinced me to put the work into *delivermail* to turn it into *sendmail.* Kurt deserves accolades for using *sendmail* when I was myself afraid to take the risk; how a person can continue to be so enthusiastic in the face of so much bitter reality is beyond me.

Kurt, Mark, Kirk McKusick, Marvin Solomon, and many others have reviewed this paper, giving considerable useful advice.

Special thanks are reserved for Mike Stonebraker at Berkeley and Bob Epstein at Britton-Lee, who both knowingly allowed me to put so much work into this project when there were so many other things I really should have been working on.

References

[Birrell82] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D., *Grapevine: An Exercise in Distributed Computing*. In *Comm. A.C.M. 25*, 4, April 82.

[Borden79] Borden, S., Gaines, R. S., and Shapiro, N. Z., *The MH Message Handling System: Users' Manual.* R-2367-PAF. Rand Corporation. October 1979.

[Crocker77a] Crocker, D. H., Vittal, J. J., Pogran, K. T., and Henderson, D. A. Jr., *Standard for the Format of ARPA Network Text Messages.* RFC 733, NIC 41952. In [Feinler78]. November 1977.

[Crocker77b] Crocker, D. H., *Framework and Functions of the MS Personal Message System.* R-2134-ARPA, Rand Corporation, Santa Monica, California. 1977.

[Crocker79] Crocker, D. H., Szurkowski, E. S., and Farber, D. J., *An Internetwork Memo Distribution Facility – MMDF.* 6th Data Communication Symposium, Asilomar. November 1979.

[Crocker82] Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages.* RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Metcalfe76] Metcalfe, R., and Boggs, D., *Ethernet: Distributed Packet Switching for Local Computer Networks, Communications of the ACM 19*, 7. July 1976.

[Feinler78] Feinler, E., and Postel, J. (eds.), *ARPANET Protocol Handbook.* NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978.

[NBS80] National Bureau of Standards, *Specification of a Draft Message Format Standard.* Report No. ICST/CBOS 80-2. October 1980.

[Neigus73] Neigus, N., *File Transfer Protocol for the ARPA Network.* RFC 542, NIC 17759. In [Feinler78]. August, 1973.

[Nowitz78a] Nowitz, D. A., and Lesk, M. E., *A Dial-Up Network of UNIX Systems.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. August, 1978.

[Nowitz78b] Nowitz, D. A., *Uucp Implementation Description.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. October, 1978.

[Postel74] Postel, J., and Neigus, N., Revised FTP Reply Codes. RFC 640, NIC 30843. In [Feinler78]. June, 1974.

[Postel77]	Postel, J., *Mail Protocol.* NIC 29588. In [Feinler78]. November 1977.
[Postel79a]	Postel, J., *Internet Message Protocol.* RFC 753, IEN 85. Network Information Center, SRI International, Menlo Park, California. March 1979.
[Postel79b]	Postel, J. B., *An Internetwork Message Structure.* In *Proceedings of the Sixth Data Communications Symposium*, IEEE. New York. November 1979.
[Postel80]	Postel, J. B., *Structured Format for Transmission of Multi-Media Documents.* RFC 767. Network Information Center, SRI International, Menlo Park, California. August 1980.
[Postel82]	Postel, J. B., *Simple Mail Transfer Protocol.* RFC821 (obsoleting RFC788). Network Information Center, SRI International, Menlo Park, California. August 1982.
[Schmidt79]	Schmidt, E., *An Introduction to the Berkeley Network.* University of California, Berkeley California. 1979.
[Shoens79]	Shoens, K., *Mail Reference Manual.* University of California, Berkeley. In UNIX Programmer's Manual, Seventh Edition, Volume 2C. December 1979.
[Sluizer81]	Sluizer, S., and Postel, J. B., *Mail Transfer Protocol.* RFC 780. Network Information Center, SRI International, Menlo Park, California. May 1981.
[Solomon81]	Solomon, M., Landweber, L., and Neuhengen, D., *The Design of the CSNET Name Server.* CS-DN-2, University of Wisconsin, Madison. November 1981. Su, Zaw-Sing, and Postel, Jon, *The Domain Naming Convention for Internet User Applications.* RFC819. Network Information Center, SRI International, Menlo Park, California. August 1982.

[UNIX83] *The UNIX Programmer's Manual, Seventh Edition,* Virtual VAX-11 Version, Volume 1. Bell Laboratories, modified by the University of California, Berkeley, California. March, 1983.

Domain Conventions

1. Introduction DC-1
2. The Structural Model DC-3
3. Advantage of Absolute Naming DC-7
4. Interoperability DC-9
5. Name Service DC-11
6. Naming Authority DC-13
7. Network-Oriented Applications DC-15
8. Mail Relaying DC-17
9. Implementation DC-19
10. Summary DC-21

A: The BNF Specification DC-23

B: An Aside on the Assignment of Simple Names DC-25

C: Further Discussion of Name Service and Name Servers DC-27

D: Further Discussion of Interoperability and Protocol Translations DC-29

1. Introduction

For many years, the naming convention ''<user>@<host>'' has served the ARPANET user community for its mail system, and the substring "<host>" has been used for other applications such as file transfer (FTP) and terminal access (Telnet). With the advent of network interconnection, this naming convention needs to be generalized to accommodate internetworking. A decision has recently been reached to replace the simple name field, ''<host>'', by a composite name field, ''<domain>''[2]. This note is an attempt to clarify this generalized naming convention, the Internet Naming Convention, and to explore the implications of its adoption for Internet name service and user applications.

The following example illustrates the changes in naming convention:

```
ARPANET Convention:    Fred@ISIF
Internet Convention:   Fred@F.ISI.ARPA
```

The intent is that the Internet names be used to form a tree-structured administrative dependent, rather than a strictly topology dependent, hierarchy. The left-to-right string of name components proceeds from the most specific to the most general, that is, the root of the tree, the administrative universe, is on the right.

The name service for realizing the Internet naming convention is assumed to be application independent. It is not a part of any particular application, but rather an independent name service serves different user applications.

2. The Structural Model

The Internet naming convention is based on the domain concept. The name of a domain consists of a concatenation of one or more <simple names>. A domain can be considered as a region of jurisdiction for name assignment and of responsibility for name-to-address translation. The set of domains forms a hierarchy.

Using a graph theory representation, this hierarchy may be modeled as a directed graph. A directed graph consists of a set of nodes and a collection of arcs, where arcs are identified by ordered pairs of distinct nodes [1]. Each node of the graph represents a domain. An ordered pair (B, A), an arc from B to A, indicates that B is a subdomain of domain A, and B is a simple name unique within A. We will refer to B as a child of A, and A a parent of B. The directed graph that best describes the naming hierarchy is called an ''in-tree'', which is a rooted tree with all arcs directed towards the root (Figure 2-1). The root of the tree represents the naming universe, ancestor of all domains. Endpoints (or leaves) of the tree are the lowest-level domains.

```
        U
      / | \
    /   |   \          U -- Naming Universe
   ^    ^    ^         I -- Intermediate Domain
   |    |    |         E -- Endpoint Domain
   I    E    I
 /   \       |
^     ^      ^
|     |      |
E     E      I
           / | \
          ^  ^  ^
          |  |  |
          E  E  E
```

Figure 2-1. The In-Tree Model for Domain Hierarchy

The simple name of a child in this model is necessarily unique within its parent domain. Since the simple name of the child's parent is unique within the child's grandparent domain, the child can be uniquely named in its grandparent domain by the concatenation of its simple name followed by its parent's simple name.

For example, if the simple name of a child is ''C1'' then no other child of the same parent may be named ''C1''. Further, if the parent of this child is named ''P1'', then ''P1'' is a unique simple name in the child's grandparent domain. Thus, the concatenation C1.P1 is unique in C1's grandparent domain.

Similarly, each element of the hierarchy is uniquely named in the universe by its complete name, the concatenation of its simple name and those for the domains along the trail leading to the naming universe.

The hierarchical structure of the Internet naming convention supports decentralization of naming authority and distribution of name service capability. We assume a naming authority and a name server associated with each domain. In Sections 5 and 6 respectively the name service and the naming authority are discussed.

Within an endpoint domain, unique names are assigned to <user> representing user mailboxes. User mailboxes may be viewed as children of their respective domains.

In reality, anomalies may exist violating the in-tree model of naming hierarchy. Overlapping domains imply multiple parentage, i.e., an entity of the naming hierarchy being a child of more than one domain. It is conceivable that ISI can be a member of the ARPA domain as well as a member of the USC domain (Figure 2-2). Such a relation constitutes an anomaly to the rule of one-connectivity between any two points of a tree. The common child and the sub-tree below it become descendants of both parent domains.

```
        U
      / | \
    /   .   \
  .     .   ARPA
.       .     | \
       USC    |   \
          \   |     .
            \ |       .
             ISI
```

Figure 2-2. Anomaly in the In-Tree Model

Some issues resulting from multiple parentage are addressed in Appendix B. The general implications of multiple parentage are a subject for further investigation.

3. Advantage of Absolute Naming

Absolute naming implies that the (complete) names are assigned with respect to a universal reference point. The advantage of absolute naming is that a name thus assigned can be universally interpreted with respect to the universal reference point. The Internet naming convention provides absolute naming with the naming universe as its universal reference point.

For relative naming, an entity is named depending upon the position of the naming entity relative to that of the named entity. A set of hosts running the ''unix'' operating system exchange mail using a method called ''uucp''. The naming convention employed by uucp is an example of relative naming. The mail recipient is typically named by a source route identifying a chain of locally known hosts linking the sender's host to the recipient's. A destination name can be, for example,

```
``alpha!beta!gamma!john''
```

where ''alpha'' is presumably known to the originating host, ''beta'' is known to ''alpha'', and so on.

The uucp mail system has demonstrated many of the problems inherent to relative naming. When the host names are only locally interpretable, routing optimization becomes impossible. A reply message may have to traverse the reverse route to the original sender in order to be forwarded to other parties.

Furthermore, if a message is forwarded by one of the original recipients or passed on as the text of another message, the frame of reference of the relative source route can be completely lost. Such relative naming schemes have severe problems for many of the uses that we depend upon in the ARPA Internet community.

4. Interoperability

To allow interoperation with a different naming convention, the names assigned by a foreign naming convention need to be accommodated. Given the autonomous nature of domains, a foreign naming environment may be incorporated as a domain anywhere in the hierarchy. Within the naming universe, the name service for a domain is provided within that domain. Thus, a foreign naming convention can be independent of the Internet naming convention. What is implied here is that no standard convention for naming needs to be imposed to allow interoperations among heterogeneous naming environments.

For example:

There might be a naming convention, say, in the FOO world, something like ''<user>%<host>%<area>''. Communications with an entity in that environment can be achieved from the Internet community by simply appending ''.FOO'' on the end of the name in that foreign convention.

```
John%ISI-Tops20-7%California.FOO
```

Another example:

One way of accommodating the ''uucp world'' described in the last section is to declare it as a foreign system. Thus, a uucp name

```
''alpha!beta!gamma!john''
```

might be known in the Internet community as

```
''alpha!beta!gamma!john.UUCP''
```

Communicating with a complex subdomain is another case which can be treated as interoperation. A complex subdomain is a domain with complex internal naming structure presumably unknown to the outside world (or the outside world does not care to be concerned with its complexity).

For the mail system application, the names embedded in the message text are often used by the destination for such purpose as to reply to the original message. Thus, the embedded names may need to be converted for the benefit of the name server in the destination environment.

Conversion of names on the boundary between heterogeneous naming environments is a complex subject. The following example illustrates some of the involved issues.

For example:

A message is sent from the Internet community to the FOO environment. It may bear the "From" and "To" fields as:

```
From: Fred@F.ISI.ARPA
To:   John%ISI-Tops20-7%California.FOO
```

where "FOO" is a domain independent of the Internet naming environment. The interface on the boundary of the two environments may be represented by a software module. We may assume this interface to be an entity of the Internet community as well as an entity of the FOO community. For the benefit of the FOO environment, the "From "and "To" fields need to be modified upon the message's arrival at the boundary. One may view naming as a separate layer of protocol, and treat conversion as a protocol translation. The matter is complicated when the message is sent to more than one destination within different naming environments; or the message is destined within an environment not sharing boundary with the originating naming environment.

While the general subject concerning conversion is beyond the scope of this note, a few questions are raised in Appendix D.

5. Name Service

Name service is a network service providing name-to-address translation. Such service may be achieved in a number of ways. For a simple networking environment, it can be accomplished with a single central database containing name-to-address correspondence for all the pertinent network entities, such as hosts.

In the case of the old ARPANET host names, a central database is duplicated in each individual host. The originating module of an application process would query the local name service (e.g., make a system call) to obtain network address for the destination host. With the proliferation of networks and an accelerating increase in the number of hosts participating in networking, the ever growing size, update frequency, and the dissemination of the central database makes this approach unmanageable.

The hierarchical structure of the Internet naming convention supports decentralization of naming authority and distribution of name service capability. It readily accommodates growth of the naming universe. It allows an arbitrary number of hierarchical layers. The addition of a new domain adds little complexity to an existing Internet system.

The name service at each domain is assumed to be provided by one or more name servers. There are two models for how a name server completes its work, these might be called ''iterative'' and ''recursive''.

For an iterative name server there may be two kinds of responses. The first kind of response is a destination address. The second kind of response is the address of another name server. If the response is a destination address, then the query is satisfied. If the response is the address of another name server, then the query must be repeated using that name server, and so on until a destination address is obtained.

For a recursive name server there is only one kind of response — a destination address. This puts an obligation on the name server to actually make the call on another name server if it can't answer the query itself.

It is noted that looping can be avoided since the names presented for translation can only be of finite concatenation. However, care should be taken in employing mechanisms such as a pointer to the next simple name for resolution.

We believe that some name servers will be recursive, but we don't believe that all will be. This means that the caller must be prepared for either type of server. Further discussion and examples of name service is given in Appendix C.

The basic name service at each domain is the translation of simple names to addresses for all of its children. However, if only this basic name service is provided, the use of complete (or fully qualified) names would be required. Such requirement can be unreasonable in practice. Thus, we propose the use of partial names in the context in which their uniqueness is preserved. By construction, naming uniqueness is preserved within the domain of a common ancestry. Thus, a partially qualified name is constructed by omitting from the complete name ancestors common to the communicating parties. When a partially qualified name leaves its context of uniqueness it must be additionally qualified.

The use of partially qualified names places a requirement on the Internet name service. To satisfy this requirement, the name service at each domain must be capable of, in addition to the basic service, resolving simple names for all of its ancestors (including itself) and their children. In Appendix B, the required distinction among simple names for such resolution is addressed.

6. Naming Authority

Associated with each domain there must be a naming authority to assign simple names and ensure proper distinction among simple names.

Note that if the use of partially qualified names is allowed in a sub-domain, the uniqueness of simple names inside that sub-domain is insufficient to avoid ambiguity with names outside the subdomain. Appendix B discusses simple name assignment in a sub-domain that would allow the use of partially qualified names without ambiguity.

Administratively, associated with each domain there is a single person (or office) called the registrar. The registrar of the naming universe specifies the top-level set of domains and designates a registrar for each of these domains. The registrar for any given domain maintains the naming authority for that domain.

7. Network-Oriented Applications

For user applications such as file transfer and terminal access, the remote host needs to be named. To be compatible with ARPANET naming convention, a host can be treated as an endpoint domain.

Many operating systems or programming language run-time environments provide functions or calls (JSYSs, SVCs, UUOs, SYSs, etc.) for standard services (e.g., time-of-day, account-of-logged-in-user, convert-number-to-string). It is likely to be very helpful if such a function or call is developed for translating a host name to an address. Indeed, several systems on the ARPANET already have such facilities for translating an ARPANET host name into an ARPANET address based on internal tables.

We recommend that this provision of a standard function or call for translating names to addresses be extended to accept names of Internet convention. This will promote a consistent interface to the users of programs involving internetwork activities. The standard facility for translating Internet names to Internet addresses should include all the mechanisms available on the host, such as checking a local table or cache of recently checked names, or consulting a name server via the Internet.

8. Mail Relaying

Relaying is a feature adopted by more and more mail systems. Relaying facilitates, among other things, interoperations between heterogeneous mail systems. The term ''relay'' is used to describe the situation where a message is routed via one or more intermediate points between the sender and the recipient. The mail relays are normally specified explicitly as relay points in the instructions for message delivery. Usually, each of the intermediate relays assume responsibility for the relayed message [3].

A point should be made on the basic difference between mail relaying and the uucp naming system. The difference is that although mail relaying with absolute naming can also be considered as a form of source routing, the names of each intermediate points and that of the destination are universally interpretable, while the host names along a source route of the uucp convention is relative and thus only locally interpretable.

The Internet naming convention explicitly allows interoperations among heterogeneous systems. This implies that the originator of a communication may name a destination which resides in a foreign system. The probability is that the destination network address may not be comprehensible to the transport system of the originator. Thus, an implicit relaying mechanism is called for at the boundary between the domains. The function of this implicit relay is the same as the explicit relay.

9. Implementation

The Actual Domains

The initial set of top-level names include:

ARPA	This represents the set of organizations involved in the Internet system through the authority of the U.S. Defense Advanced Research Projects Agency. This includes all the research and development hosts on the ARPANET and hosts on many other nets as well. But note very carefully that the top-level domain ''ARPA'' does not map one-to-one with the ARPANET — domains are administrative, not topological.

Transition

In the transition from the ARPANET naming convention to the Internet naming convention, a host name may be used as a simple name for an endpoint domain. Thus, if "USC-ISIF" is an ARPANET host name, then ''USC-ISIF.ARPA'' is the name of an Internet domain.

10. Summary

A hierarchical naming convention based on the domain concept has been adopted by the Internet community. It is an absolute naming convention defined along administrative rather than topological boundaries. This naming convention is adaptive for interoperations with other naming conventions. Thus, no standard convention needs to be imposed for interoperations among heterogeneous naming environments.

This Internet naming convention allows distributed name service and naming authority functions at each domain. We have specified these functions required at each domain. Also discussed are implications on network-oriented applications, mail systems, and administrative aspects of this convention.

Appendix A: The BNF Specification

We present here a rather detailed ''BNF'' definition of the allowed form for a computer mail ''mailbox'' composed of a ''local-part'' and a ''domain'' (separated by an at sign). Clearly, the domain can be used separately in other network-oriented applications.

```
  <mailbox> ::= <local-part> "@" <domain>

  <local-part> ::= <string> | <quoted-string>

  <string> ::= <char> | <char> <string>

  <quoted-string> ::=  """ <qtext> """

  <qtext> ::=  "\" <x> | "\" <x> <qtext> | <q> | <q> <qtext>

  <char> ::= <c> | "\" <x>

  <domain> ::= <naming-domain> | <naming-domain> "." <domain>

  <naming-domain> ::=  <simple-name> | <address>

  <simple-name> ::= <a> <ldh-str> <let-dig>

  <ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

  <let-dig> ::= <a> | <d>

  <let-dig-hyp> ::= <a> | <d> | "-"

  <address> :: =  "#" <number> | "[" <dotnum> "]"

  <number> ::= <d> | <d> <number>
```

<dotnum> ::= <snum> "." <snum> "." <snum> "." <snum>

<snum> ::= one, two, or three digits representing a decimal integer value in the range 0 through 255

<a> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case

<c> ::= any one of the 128 ASCII characters except <s> or <SP>

<d> ::= any one of the ten digits 0 through 9

<q> ::= any one of the 128 ASCII characters except CR, LF, quote ("), or backslash (\)

<x> ::= any one of the 128 ASCII characters (no exceptions)

<s> ::= "<", ">", "(", ")", "[", "]", "\", ".", ",", ";", ":", "@", """", and the control characters (ASCII codes 0 through 31 inclusive and 127)

Note that the backslash, "\", is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe\,Smith" could be used to indicate a single nine character user field with comma being the fourth character of the field.

The simple names that make up a domain may contain both upper and lower case letters (as well as digits and hyphen), but these names are not case sensitive.

Hosts are generally known by names. Sometimes a host is not known to the translation function and communication is blocked. To bypass this barrier two forms of addresses are also allowed for host ‘‘names’’. One form is a decimal integer prefixed by a pound sign, (#). Another form, called "dotted decimal", is four small decimal integers separated by dots and enclosed by brackets, e.g., "[123.255.37.2]", which indicates a 32-bit ARPA Internet Address in four 8-bit fields. (Of course, these numeric address forms are specific to the Internet, other forms may have to be provided if this problem arises in other transport systems.)

Appendix B: An Aside on the Assignment of Simple Names

In the following example, there are two naming hierarchies joining at the naming universe 'U'. One consists of domains (S, R, N, J, P, Q, B, A); and the other (L, E, F, G, H, D, C, K, B, A). Domain B is assumed to have multiple parentage as shown.

```
                  U
                /   \
              /       \
            J           L
          /               \
        N                   E
      /   \               /   \
    R       P           D       F
  /           \         | \      \
S               Q       C  (X)     G
                  \   /   \          \
                    B       K          H
                  /
                A
```

Figure B-1. Illustration of Requirements for the Distinction of Simple Names

Suppose someone at A tries to initiate communication with destination H. The fully qualified destination name would be

```
H.G.F.E.L.U
```

Omitting common ancestors, the partially qualified name for the destination would be

```
H.G.F
```

To permit the case of partially qualified names, name server at A needs to resolve the simple name F, i.e., F needs to be distinct from all the other simple names in its database.

To enable the name server of a domain to resolve simple names, a simple name for a child needs to be assigned distinct from those of all of its ancestors and their immediate children. However, such distinction would not be sufficient to allow simple name resolution at lower-level domains because lower-level domains could have multiple parentage below the level of this domain.

In the example above, let us assume that a name is to be assigned to a new domain X by D. To allow name server at D to resolve simple names, the name for X must be distinct from L, E, D, C, F, and J. However, allowing A to resolve simple names, X needs to be also distinct from A, B, K, as well as from Q, P, N, and R.

The following observations can be made.

- Simple names along parallel trails (distinct trails leading from one domain to the naming universe) must be distinct, e.g., N must be distinct from E for B or A to properly resolve simple names.
- No universal uniqueness of simple names is called for, e.g., the simple name S does not have to be distinct from that of E, F, G, H, D, C, K, Q, B, or A.
- The lower the level at which a domain occurs, the more immune it is to the requirement of naming uniqueness.

To satisfy the required distinction of simple names for proper resolution at all levels, a naming authority needs to ensure the simple name to be assigned distinct from those in the name server databases at the endpoint naming domains within its domain. As an example, for D to assign a simple name for X, it would need to consult databases at A and K. It is, however, acceptable to have simple names under domain A identical with those under K. Failure of such distinct assignment of simple names by naming authority of one domain would jeopardize the capability of simple name resolution for entities within the subtree under that domain.

Appendix C: Further Discussion of Name Service and Name Servers

The name service on a system should appear to the programmer of an application program simply as a system call or library subroutine. Within that call or subroutine there may be several types of methods for resolving the name string into an address.

- First, a local table may be consulted. This table may be a complete table and may be updated frequently, or it may simply be a cache of the few latest name to address mappings recently determined.
- Second, a call may be made to a name server to resolve the string into a destination address.
- Third, a call may be made to a name server to resolve the string into a relay address.

Whenever a name server is called it may be a recursive server or an interactive server.

- If the server is recursive, the caller won't be able to tell if the server itself had the information to resolve the query or called another server recursively (except perhaps for the time it takes).
- If the server is iterative, the caller must be prepared for either the answer to its query, or a response indicating that it should call on a different server.

It should be noted that the main name service discussed in this memo is simply a name string to address service. For some applications there may be other services needed.

- For example, even within the Internet there are several procedures or protocols for actually transferring mail. One need is to determine which mail procedures a destination host can use. Another need is to determine the name of a relay host if the source and destination hosts do not have a common mail procedure. These more specialized services must be specific to each application since the answers may be application dependent, but the basic name to address translation is application independent.

Appendix D: Further Discussion of Interoperability and Protocol Translations

The translation of protocols from one system to another is often quite difficult. Following are some questions that stem from considering the translations of addresses between mail systems:

- What is the impact of different addressing environments (i.e., environments of different address formats)?
- It is noted that the boundary of naming environment may or may not coincide with the boundary of different mail systems. Should the conversion of naming be independent of the application system?
- The boundary between different addressing environments may or may not coincide with that of different naming environments or application systems. Some generic approach appears to be necessary.
- If the conversion of naming is to be independent of the application system, some form of interaction appears necessary between the interface module of naming conversion with some application level functions, such as the parsing and modification of message text.
- To accommodate encryption, conversion may not be desirable at all. What then can be an alternative to conversion?

Glossary

address	An address is a numerical identifier for the topological location of the named entity.
name	A name is an alphanumeric identifier associated with the named entity. For unique identification, a name needs to be unique in the context in which the name is used. A name can be mapped to an address.
complete (fully qualified) name	A complete name is a concatenation of simple names representing the hierarchical relation of the named with respect to the naming universe, that is it is the concatenation of the simple names of the domain structure tree nodes starting with its own name and ending with the top level node name. It is a unique name in the naming universe.
partially qualified name	A partially qualified name is an abbreviation of the complete name omitting simple names of the common ancestors of the communicating parties.
simple name	A simple name is an alphanumeric identifier unique only within its parent domain.
domain	A domain defines a region of jurisdiction for name assignment and of responsibility for name-to-address translation.
naming universe	Naming universe is the ancestor of all network entities.
naming environment	A networking environment employing a specific naming convention.

name service	Name service is a network service for name-to-address mapping.
name server	A name server is a network mechanism (e.g., a process) realizing the function of name service.
naming authority	Naming authority is an administrative entity having the authority for assigning simple names and responsibility for resolving naming conflict.
parallel relations	A network entity may have one or more hierarchical relations with respect to the naming universe. Such multiple relations are parallel relations to each other.
multiple parentage	A network entity has multiple parentage when it is assigned a simple name by more than one naming domain.

REFERENCES

1. F. Harary, ''Graph Theory'', Addison-Wesley, Reading, Massachusetts, 1969.
2. J. Postel, ''Computer Mail Meeting Notes'', RFC-805, USC/Information Sciences Institute, 8 February 1982.
3. J. Postel, ''Simple Mail Transfer Protocol'', RFC-821, USC/Information Sciences Institute, August 1982.
4. D. Crocker, ''Standard for the Format of ARPA Internet Text Messages'', RFC-822, Department of Electrical Engineering, University of Delaware, August 1982.