Mail Service Administration

Chapter 12
Administering the Mail Service

Introduction to Mail Service Administration

The mail service provides electronic communications between users on the same computer or between computers connected together on a network using the Simple Mail Transfer Protocol (SMTP). It supports two addressing schemes, known as ''bang'' style and ''domain'' style.

> **NOTE:** You do not need to do anything for mail to work in the default manner. That is, users on your system can send each other mail and if your system has some type of networking, your users can send mail to users on any of the machines that your system may be connected to. Examining the `/etc/mail/mailflgs` and `/etc/mail/mailsurr` files will give you a sense of what type of connectivity is available on your system. See the section Before You Start for more information.

You can take advantage of various options within the mail service that enable you to do any of the following:

- establish a domain name
- set up a smarter host
- administer a set of sites so they all send mail with the same system name
- set up the mail directory to be shared across a networked file system, such as NFS
- fill in alias information
- set up a connection to another site that uses the SMTP

This chapter provides information on these options.

Command Line vs. Desktop

If you are using the UnixWare Desktop, the options listed in the previous section plus additional extended options such as deleting empty mail files and enabling a failsafe mail forwarding mechanism are available from the `Mail_Setup` window. See the chapter *Setting Up and Configuring Mail* in the book System Owner Handbook for details.

The `Mail_Setup` interface of the desktop provides customized mail administration files [see the next section Mail Administration Files]. From the command line, you need to invoke certain commands to make the desired changes to these files.

Mail Administration Files

There are several files that are important to mail administration.

- The `/usr/lib/mail/mailsurr.proto` file contains the prototype surrogate commands that determine how mail are delivered to the intended destination.. This file serves as input to the `/usr/lib/mail/surrcmd/createSurr` command and, together with the `/etc/mail/mailflgs` file, determines the generated `mailsurr` file.

The surrogate file `/etc/mail/mailsurr`, which is generated by the `createSurr` command, describes how to rewrite addresses and how to deliver messages through the networks. [See mailsurr(4) in the System Files and Devices Reference.]

> **NOTE:** Make all your changes in the `mailsurr.proto` file, not in the `mailsurr` file. Once you have made the changes, invoke `createSurr` to create the necessary `mailsurr` file.
>
> Manipulation of the `mailsurr.proto` and subsequent invocation of `createSurr` are transparent when you use the `Mail_Setup` from the graphical user interface. See the chapter *Setting Up and Configuring Mail* in the book System Owner Handbook for more information.

- The `/etc/mail/mailflgs` file contains a set of flags defining the system configuration. This file tells you if you have UUCP, Domain Name Service (DNS), Network Information Service (NIS), and others configured on your system. [See mailflgs(4) in the System Files and Devices Reference.]
- The configuration file, `/etc/mail/mailcnfg`, which is optional, permits various per−site options to be established. This file is not delivered with the mail service: it gets created the first time you use the `Mail_Setup` from the desktop or you must create it yourself if you don't have the desktop and you want to customize your environment using the available set of options. [See mailcnfg(4) in the System Files and Devices Reference.]
- The master alias path file, `/etc/mail/namefiles`, by default points to one alias file, `/etc/mail/names`, and one alias directory, `/etc/mail/lists`. Directories and files may be added to the list. These files and directories define name mappings and address lists. [Both files are described in mailalias(1) in the Command Reference.]

Mail Addressing Styles

The default surrogate file contains command entries for translating bang−style addresses to domain−style addresses. Bang−style addressing is characterized by exclamation points (or ''bangs'') within the address and looks like *host*`!`*user* or *host1*`!`*host2*`!`*user*. Domain−style addressing is characterized by the commercial at sign (@) and looks like *user*`@`*host*`.`*domain* or *user*`@`*host*.

Before You Start

To determine your current mail configuration, examine your system's `/etc/mail/mailflgs` file.

> **NOTE:** If the `mailflgs` file does not exist or has no entry, type `/usr/lib/mail/surrcmd/createSurr` on the command line. This creates/populates the `mailflgs` file with the appropriate flags defining the system configuration.

Some of the flags that you may find in the file and their meanings are:

`UUCP`

There is at least one entry in the `/etc/uucp/Systems` file. Users on your system may be able to send mail to users of the systems defined in this file via UUCP. See the chapter Administering the Basic Networking Utilities in the book *Network Administration* for more information on UUCP.

`DNS`

The Domain Name Service resolver is configured on the system. Entries in the `/etc/resolv.conf` file define your system domain name and name servers. See the chapter Using Domain Name Service with TCP/IP in the book *Network Administration* for more information on DNS.

The `SMARTERHOST` variable is present in `/etc/mail/mailcnfg` file. See the section Establishing a

Smarter Host for more information.

`MSH`

The presence of this flag indicates that your system is configured as a Message Handling Service (MHS) gateway. See the chapter *Setting Up and Configuring Mail* in the book System Owner Handbook for more information on how to configure your system as an MHS gateway.

Other flags may be present. See the mailflgs(4) manual page for descriptions of other defined flags.

Routing Mail Messages Over a Network

The ability to send mail to different systems depends primarily on their connectivity and the established addresses.

Establishing Domain Addresses

The mail system, as distributed, recognizes two forms of domain style addresses:

`user@host` *`user@host`*`.UUCP`

It does not know about

`user@host.domain`

A domain name is an internationally recognized and registered name for a set of computers. Commercial entities may be registered under domain names similar to `.`*`company-name`*`.COM` and educational entities may be registered under domain names similar to `.`*`school`*`.EDU`.

NOTE: `.UUCP` is not a true domain name. The high–level domain names of `.COM` and `.EDU` are assigned by a central authority called InterNIC (Internet Network Information Center).

A system generally knows how to establish direct connections to other computers within the local domain, but you may want to use a smarter host to take care of other domains.

To establish the local Network Information Service (NIS) domain name every time you boot your system, type

`/usr/bin/domainname` *`domain`*

where *`domain`* is replaced with the domain name, such as `.`*`company-name`*`.COM` (or whatever is appropriate) and contains a leading period. (Preceding backslashes are added to any periods in the domain name, changing ‘‘`.`’’ to ‘‘`\.`,’’ before being passed to the regular expressions in the surrogate file.)

The Domain Name Service (DNS) is established in the `/etc/resolv.conf` file.

The domain name may also be established using the `DOMAIN` variable of the `/etc/mail/mailcnfg` file.

Establishing a Smarter Host

Although it is possible to maintain the data files for the entire network so the system knows about hundreds or thousands of other systems that can be contacted, it is impractical to do so. It is often much easier to set up what is known as a ‘‘smarter host,’’ that is, a reference to a UNIX system where remote mail will be shipped if the mail destination is not known to the local computer. For example, assume you need to send a mail message to `hosta!tony`, but your local computer does not know about `hosta`. The mail message can be automatically routed to the computer `worldly`, which has a more extensive list of network connections, if `worldly` is set up as a smarter host.

Procedure

To establish a smarter host, do the following:

1. Add a line to `/etc/mail/mailcnfg` that says

    `SMARTERHOST=`*`smhost`*

    where *`smhost`* is replaced with the name of the smarter system (`worldly`).

    NOTE: `SMARTERHOST` is usually set to the mail gateway machine listed in the `/etc/hosts` file.

2. Enter

    `/usr/lib/mail/surrcmd/createSurr`

    on the command line.

Establishing a Mail Cluster or Gateway

With the arrival of inexpensive personal computers, it is often desirable to assign a single name to a set (or a cluster) of computers by which all the computers in the cluster are known to external computers, for purposes of mail delivery. For example, a cluster of computers known internally under names such as `Xsysa`, `Xsysb`, and `Xsysc` could be assigned the cluster name of `Xsys`. Mail sent from any of these computers would be shown as being from `Xsys`; that is, the internal names would not be known outside the cluster.

Procedure

To establish a cluster name, do the following:

1. Add a line such as the following to `/etc/mail/mailcnfg`:

    `CLUSTER=`*`extname`*

    where *`extname`* is the name by which the computer is known externally (such as `Xsys`). In the surrogate file, `/etc/mail/mailsurr`, you can then use `%L` to refer to the value of `CLUSTER`.

2. Enter

    `/usr/lib/mail/surrcmd/createSurr`

    on the command line.

Establishing Mail Service on a Networked File System

With the arrival of inexpensive local area network (LAN) and networked file systems (such as NFS), it is often desirable to set up clusters of computers that share file systems.

It is also possible to share `/var/mail` across the computers.

If you choose to establish a mail cluster with a shared `/var/mail` you can arrange to have all users' mailboxes created on only one computer but accessible from all computers.

As an example, assume you want the systems `Xsysa`, `Xsysb`, and `Xsysc` to share a single mail directory under `Xsysa`. In addition, the entire file system for each system is mounted under the names `/Xsysa`, `/Xsysb`, and `/Xsysc`. All users have home directories under file systems named `/homea`, `/homeb`, and `/homec`, which are mounted on the corresponding computers.

Procedure

To establish mail service on a networked file system given the previous scenario, complete the following steps:

1. Make sure `/var/mail` from `Xsysa` is shared. Use the `dfshares` command to find out. From `Sysa`, enter:

   ```
   dfshares
   ```

   From `Xsysb` and `Xsysc`, enter:

   ```
   dfshares Xsysa
   ```

   Both commands should give:

   ```
   Xsysa:/var/mail
   ```

2. Remove the directory `/var/mail/:saved` from systems that will not have a local `/var/mail` (`Xsysb` and `Xsysc`).

3. On systems `Xsysb` and `Xsysc`, add a line to `/etc/mail/mailcnfg` that says

   ```
   FAILSAFE=Xsysa
   ```

   With this specified, `mail` looks for the presence of `/var/mail/:saved`. If the directory is not there (an indication that the network connection to `Xsysa` has been lost), `mail` requeues the file to be delivered to `Xsysa` via other means (such as by UUCP or SMTP).

4. Move any mailboxes from `/var/mail` on `Xsysb` and `Xsysc` to a temporary holding spot in `Xsysa`. (If you don't, the files will be inaccessible.)

5. On systems `Xsysb` and `Xsysc`, mount `/var/mail` from `Xsysa`.

6. Create, on all systems, a file named `/etc/mail/notify.sys` with contents similar to the following:

   ```
   Xsysa /Xsysa
   Xsysb /Xsysb
   Xsysc /Xsysc
   ```

   The first column lists the name of the system; the second gives the path of the `root` file system for each computer.

   This file allows the `notify` program to identify where the user is logged on (thus, users are notified when new mail arrives),

7. Create, on all systems, a file named `/etc/mail/notify.fsys` with contents similar to the following:

   ```
   /homea Xsysa
   /homeb Xsysb
   /homec Xsysc
   ```

   The first column gives the path of the file system; the second contains the name of the system on which that file system is normally mounted.

   This file allows the `notify` program to handle a network failure. If `notify` cannot open the mail file for writing, it looks up the file system in this list and requeues the file to be delivered to the corresponding system via other means (such as by UUCP or SMTP).

Administering Alias Lists

Before delivering a local mail message, `mail` looks up the user name to see if the user name has been assigned an alias. An alias may reference simply another name or a list of names.

The master alias path file (`/etc/mail/namefiles`) contains a list of files that `mail` searches for aliases. As distributed, this list contains the file `/etc/mail/names` and the directory `/etc/mail/lists`, to be searched for aliases. `/etc/mail/names` may contain mail forwarding aliases of the form:

```
alias mail-address
```

where *alias* is any arbitrary string that can be used in place of the actual address specified by *mail-address* for sending mail. The `/etc/mail/lists` directory may contain files whose names represent the aliases and the contents of each specifying the actual mail address or addresses. For example, in the `lists` directory, you may create the file `clinton` which contains the line `whitehouse!president`. You may then send your mail to the president using the alias `clinton`. To implement this using the `names` file, you simply add a line as follows:

```
clinton  whitehouse!president
```

If a named alias is found at the beginning of a line within an alias file, the rest of that line is used as the alias. This may contain a single name or a list of names separated by blank spaces. For example, if you want to set up a group mailing list (such as `andy.group`) that is to be expanded, add a line similar to the following to the alias file:

```
andy.group tony paul john ned gary hailey mike
```

Recursive references are permitted, as in this reference to `andy.group` within another alias:

```
armida.dept andy.group danielle.group bob.group \
 lee.group pier.group
```

Several alias files can be listed in `namefiles`, which may be kept anywhere on the computer. This permits different alias files to be owned by different administrators.

NIS and DNS Support

If the Network Information Service (NIS) or Domain Name Service (DNS) is available, the mail system will also use that service to locate addressing information not found locally. See the chapter Using Domain Name Service with TCP/IP in the book *Network Administration* for more information on DNS and part NIS Administration in the book *Network Administration* for more information on NIS.

Using the mailsurr File

`/etc/mail/mailsurr` contains the surrogate commands that dictate how mail is routed and transported to/from your system. Mail surrogates include

- logging the mail traffic going through the system
- using a path translation program that gives the shortest route to a given system
- controlling mail resource access

These are described in the following sections. See the mailsurr(4) manual page for a complete description of surrogate files.

> **NOTE:** Make your changes in the `/etc/lib/mail/mailsurr.proto` file; then invoke the `/usr/lib/mail/surrcmd/createSurr` to implement your changes in the `/etc/mail/mailsurr` file.

Logging Mail

It may be necessary to keep a log of traffic going through the system. For example, if you were to write a program called `/usr/lib/mail/surrcmd/logmail` that takes three arguments (the names of a log file, a sender, and a recipient), it could log all external mail flowing through the system by using this surrogate entry:

```
'.+!.+' '.*' '> logmail W=1;B=*; /var/adm/mailtransport %R' '%n'
```

Because the path used for surrogate commands includes `/usr/lib/mail/surrcmd`, there is no need to use a full path to reference `logmail`.

Another example would be to log traffic to or from a particular system. In the following example, we're logging traffic from our system to system `xyz` and traffic from `abc` to our system:

```
'.*' 'xyz!.+' '> W=1;B=*; logmail /var/adm/mailto-xyz %R' '%n'
'abc!.+' '.*' '> W=1;B=*; logmail /var/adm/mailfrom-abc %R' '%n'
```

The `W=1;` indicates that `mail` will not wait for the logging program to complete before continuing its work. The `B=*;` indicates that `mail` will execute a single call to `logmail`, passing it a list of all recipients to whom mail was delivered.

Using pathrouter for Smart Routing

Many systems use a path translation program that provides the shortest route to a given system, based on various criteria or a database. The `pathrouter` program is an example of a path translation program and is typically used by a machine that acts as a mail gateway to a group of machines. As an alternative to using a smarter host, the `pathrouter` program can be invoked as a final step in the `mailsurr` file.

The `pathrouter` program uses the file `/etc/uucp/paths` to decide how to rewrite addresses. The following `paths` file is a sample setup for a gateway machine. Each entry prefixes the destination routes with an exclamation mark (`!`) to reduce the number of translations done within the `mailsurr` file. The following example rewrites the domain addresses `defgmbh.ge` and any domain address based in the `uk` domain to be routed via the `abcinc` machine. The `xyzinc.com` domain address is converted to `!xyzinc!%s` to use the UUCP transport rather than SMTP.

```
.uk           !abcinc!%s  0
.xyzinc.com   !xyzinc!%s  0
abcinc        !abcinc!%s  0
defgmbh.ge    !abcinc!defgmbh.ge!%s  0
windsor       !abcinc!windsor!%s  0
xyzinc        !xyzinc!%s  0
```

In this example, a smarter host is not defined, and so any addresses that are not matched by `pathrouter` are left unchanged. The SMTP DNS lookup will resolve the address and perform delivery over SMTP, and mail to any unknown address will be returned.

If the `/etc/uucp/paths` file exists when you invoke the `createSurr` command, the following entry is included in the resulting `/etc/mail/mailsurr` file:

```
'.+' '[^@]+' 'Translate R=|/usr/lib/mail/surrcmd/pathrouter %n'
```

This entry provides smart routing to your mail system.

See pathrouter(1M) and paths(4) for more information on smart routing capability.

Controlling Mail Resource Access

It is sometimes necessary to control access to commercial services. You can prevent any nonlocal users from sending mail to the commercial site using the `Accept` and `Deny` commands:

```
'[^!]+'  'commail!.+' 'Accept'
'.+'     'commail!.+' 'Deny'
```

You can also use an external program to check the sender's path to see if it is a valid user of the service. For example, the following shell script returns `0` if the sender is a valid system, and `1` otherwise:

```
case "$1" in
    abc | def | ghi ) exit 0 ;;
    * ) echo "$1 is not permitted to send mail to external service"
 exit 1 ;;
esac
```

If the script were installed as `/usr/lib/mail/surrcmd/chksender`, it would be invoked as a delivery agent that either continues or fails:

```
# check senders more than one hop away
'.+!(.+)![^!]+' 'commail!.+' '< C=0;F=*; chksender \\1'
# check senders one hop away
'(.+)![^!]+'    'commail!.+' '< C=0;F=*; chksender \\1'
```

Establishing SMTP Service

The Simple Mail Transfer Protocol (SMTP) mail service is a group of programs that allows UnixWare system mail to send and receive mail. This protocol is typically used over TCP/IP networks. However, the SMTP processes can connect to a remote SMTP server over any TLI or XTI–based, connection–oriented transport medium that has been administered to have an SMTP service.

Procedure

To establish SMTP service, do the following:

1. Make sure that the Internet Utilities package (`inet`) from the Networking Set is installed. To check, enter:

   ```
   # pkginfo inet
   ```

   This should return:

   ```
   system      inet            Internet Utilities
   ```

   If not, you need to install the Internet Utilities package.

2. Invoke the `createSurr` command:

   ```
   # /usr/lib/mail/surrcmd/createSurr
   ```

3. After execution of the command, the mail surrogate file `/etc/mail/mailsurr` should have the entry that establishes a call to the SMTP queuer program (`smtpqer`).

Mail is addressed using the standard UNIX system mail format of *host*`!`*user* or *user*`@`*host*. If *host* is known to support SMTP mail delivery, the mail is queued for delivery using SMTP. If not, `smtpqer` does not accept the message, and delivery is done by subsequent surrogates in the `mailsurr` file.

All messages spooled for SMTP delivery are stored in the directory `/var/spool/mailq`.

The list of computers that will accept SMTP mail is determined by querying `netdir_getbyname`. See netdir(3N) in the Operating System API Reference and netconfig(4) in the System Files and Devices Reference for further information.

By default, the SMTP daemon `smtpd` always starts when your system is booted. If `smtpd` finds that there are no networks installed for which the SMTP service is defined, it exits.

When `smtpd` receives a piece of mail, it does two things:

- It adds a `Received: from` line at the beginning of the mail message;
- It hands the message to `rmail` for delivery.

Setting Up SMTP to Listen over Multiple Networks

`smtpd` listens to any connection–oriented TLI or XTI network that provides the SMTP service. TLI/XTI networks are specified in `/etc/netconfig`. For each network that is connection–oriented, `smtpd` uses `netdir_getbyname` to determine if the SMTP service exists for that network. If the service does exist, a port is opened at the address returned by this function. To make `smtpd` listen to a new network, do the following:

1. Add the new network to the `/etc/netconfig` file. See netconfig(4) for more information on the syntax of this configuration file.
2. Restart `smtpd` by killing the `smtpd` process and invoking it again.

```
# kill process-id
# /usr/lib/mail/surrcmd/smtpd
```

`process-id` is the id of the `smtpd` process.

Modifying the Mail Header

The surrogate commands in the `/etc/mail/mailsurr` file control how mail addresses are interpreted and delivered. You can also manipulate the `mailsurr` file (through the `mailsurr.proto` file) to modify the header of the mail message. That is the purpose of the `/etc/mail/rewrite` file and the `mailR` language. As a small example of what can be done in the `rewrite` file, consider this small function:

```
function main()
{
 delete_headers("Foo-Bar");
 if (!exists("Date"))
  prepend_header("Date", "To", rfc822date());
}
```

This function deletes all `Foo-Bar` headers in the message, including those that are spelled `FOO-BAR`, `foo-bar`, `FoO-bAr`, or any other similar header. It then checks to see if the message has a `Date` header and, if not, creates one. The new header will be placed in front of any existing `To` headers and will have the value returned by the function `rfc822date()`. If no `To` header exists, the new header will be placed at the beginning of the headers.

The /etc/mail/rewrite File

The `rewrite` file controls how the mail message is presented to the intended receiver. `rewrite` contains header rewriting rules using the `mailR` language. It consists of a series of function definitions, including `main` and `local`. The function `main` is executed before the rules in the surrogate file `mailsurr` are executed. The function `local` is executed before a mail message is delivered locally.

The mailR Language

The `mailR` language is a full−featured language, with a variety of control structures and variables. It uses a mail message as its input, generating another mail message as its output. It is similar in nature to awk(1) in that variables are automatically converted from strings to doubles as needed, and arrays are indexed by any arbitrary string (associative arrays). However, unlike `awk`

- variables must be declared before being used.
- there are no global variables.
- all statements must end with a semi−colon.
- string concatenation is performed using the `@` operator.
- function arguments can be passed by value or by reference.
- `for` statements can loop over the values of arrays as well as the indexes of arrays.
- special functions, known as generators, are available.

Example: /etc/mail/rewrite File

Now let's look at a more extensive example:

```
# call rewrite to begin header modification
function main()
{
    rewrite("To");  # modify To: headers
    rewrite("Cc");  # modify Cc: headers
    rewrite("Bcc");  # modify Bcc: headers
    rewrite("From"); # modify From: header
}
# rewrite address headers by guaranteeing that a
# domain exists and by deleting sub-domain information
function rewrite(var hdr_name)
{
    var old_Hdr;
    # loop through all headers of the given name
    for (old_Hdr from headers(hdr_name))
 {
 var hdrval, new_Hdr, address_count = 0;
 # loop through all addresses in the header
 for (hdrval from addrparse(old_Hdr))
     {
     var old_domain = hdrval["domain_part"], new_domain;
     # if no domain exists, tack on our domain
     if (!old_domain)
  {
  old_domain = domain();
  # convert !-style addresses to domain-style addresses
  if (hdrval["local_part"] ~ "!")
      {
      var new_address =
   bangtodomain(hdrval["address"] @ old_domain, "@r!l%r");
      var new_hdrval = addrparse(new_address)[0];
      hdrval["route"] = new_hdrval["route"];
      hdrval["local_part"] = new_hdrval["local_part"];
```

```
        old_domain = hdrval["domain_part"] =
     new_hdrval["domain_part"];
        }
    }
       # does the name say xyz.foo.com?
       if (old_domain ~ ".+\\\.foo\.com")
    new_domain = "foo.com";
       # does the name say xyz.bar.com?
       else if (old_domain ~ ".+\\\.bar\.com")
    new_domain = "bar.com";
       # use the domain information as it stands
       else
    new_domain = old_domain;
       # if any group name information exists, preserve it
       if (hdrval["group_name"])
    new_Hdr[address_count] = hdrval["group_name"] @ ": " @
        hdrval["route"] @ hdrval["local_part"] @
        new_domain @ " " @ hdrval["comment"] @
        hdrval["errors"] @ "; ";
       else
    new_Hdr[address_count] =
        hdrval["route"] @ hdrval["local_part"] @
        new_domain @ " " @ hdrval["comment"] @
        hdrval["errors"];
       address_count = address_count + 1;
       }
   # Replace the value of the old header with the new header.
   # Separate addresses with commas and put each address on
   # a line by itself.
   old_Hdr = "";
   if (address_count > 0)
       {
       var j;
       old_Hdr = new_Hdr[0];
       for (j = 1; j < address_count; j = j + 1)
    old_Hdr = ",\n" @ new_Hdr[j];
       }
   else
       old_Hdr = new_Hdr[0];
   }
  }
```

This example begins with the `main()` function, which calls the function `rewrite()` to modify, in turn, the `To`, `Cc`, `Bcc`, and `From` headers. The `rewrite()` function starts out by looping through all of the headers of the given type, using the built–in function `headers()`. The `for` loop is executed once for each header, with the variable *`old_hdr`* set as a reference to each value in turn. The list of addresses from each header is then parsed (into its separate pieces) using the built–in function `addrparse()`. The domain portion of the address is then examined to see if a value is present. If not, the current domain is used as the setting for `domain`.

The address is then checked to see if it contains a bang–style address. If it does, it is converted to domain–style, using the built–in function `bangtodomain()` and reparsed; the reparsed address is then assigned as the mail address.

The domains are examined to see if they contain subdomains (of the domains `foo.com` or `bar.com`), and, if so, the domain is changed to the upper–level domain. The address is recreated, along with any RFC 822 group names, the comments, and any errors which may have been detected during the parsing. The new address is stored into the array new_Hdr.

After all of the addresses have been parsed, the header's value is changed by looping over the array new_Hdr and putting the addresses together (separated by commas and newlines). Each continuation line of a header must begin with white space. The mailR language oversees this requirement and guarantees that it is done.

Many other built–in mailR functions are available, each described in the mailR(5) manual page. It is also possible to pipe the entire message or the body of the message through an external command, instead of only modifying the headers of the message.

The mail command automatically calls the mailR function main() before doing any parsing of the mail surrogate file. Before the message is delivered to a local mailbox, the mailR function local() is invoked. It is also possible to invoke a mailR function just prior to the message being passed to a delivery command. For example, the surrogate entry

```
'.+'   'xxgate!.+'   '< H=rewrite_for_xxgate;B=1024;
    uux -a %R -p -- \\1!rmail' '(\\2)'
```

uses uux to send the message to the system xxgate, but before doing so, rewrites it according to the mailR function rewrite_for_xxgate(). This might be useful in cases where there are special requirements for mail coming into the system xxgate, such as requiring bang–style addresses in all To: headers.

Note that this rewriting does not affect the message as seen by other recipients. Sometimes, however, it is necessary to have a message rewritten based on who it is coming from or going to, but irrespective of the delivery mechanism. This is accomplished using a Rewrite surrogate entry, as in

```
'xxgate!.+'   '.+'   'Rewrite rewrite_coming_from_xxgate'
```

This says that any messages coming from the system xxgate are to be rewritten according to the mailR function rewrite_coming_from_xxgate(). This might be useful if you know that messages coming from the system xxgate have certain idiosyncrasies, such as using a special format for encoding attachments within the message.

See the file /etc/mail/rewrite.samp for other examples of mailR functions.

The localmail Command

Consider a user population spread across a group of machines (sysa, sysb and sysc) that has a disjoint address space. That is, a given user only has a login on one of the machines. Sending mail to a user on one of those machines requires knowing which machine the user resides on. However, you can create a flat name space across the entire set of machines, such that you can send mail to user xyz on any of the machines and have the mail go to the user on the proper machine. This can be done in several different ways:

- Set up accounts on all machines for all users, and establish forwarding files for each user so the mail is sent to the proper machine. This quickly becomes an administrative headache.

- Create a mailing list that will be placed on each machine which lists the user and the system on which that user belongs. For example, such a mailing list might read

  ```
  usera   usera@sysb
  userb   userb@sysa
  userc   userc@sysb
  userd   userd@sysc
  ```

  This way, mail to a given user will stay on the machine if it belongs there, and go to the proper machine if it isn't local. However, this also requires that whenever a user is added to any of the systems, the mailing list must be updated on all machines.

- Alternatively, the localmail program may be used. It takes a user name and looks it up on the system to see if it is to be considered a local user name. If it is local, it just prints the user name; but if not, it rewrites the user name according the options passed to it.

This makes `localmail` ideal for use as a translation surrogate. For example, the following surrogate entry could be placed at the end of the surrogate file:

```
&'.+'   '[^!@]+'   'Translate T=1;B=*; R=|localmail -p -S @sysa' '%n'
```

Any user names that are not local will be returned with the suffix `@sysa` added, which causes the mail to be forwarded to the system `sysa`. That single system would then need a master mailing list, as described, which specifies where the user actually lives.

This is similar to how a smarter host is handled. With a smarter host, any remote address that is not recognized is rewritten to go to another host, which presumably knows how to handle such addresses. In this case as well, user names that are not recognized as local user names are rewritten to go to another host, which presumably knows how to handle such addresses.

See localmail(1M) for more information on the `localmail` command.

Support for the Multipurpose Internet Mail Extensions (MIME)

The introduction to the MIME standard says:

> RFC 822 defines a message representation protocol which specifies considerable detail about message headers, but which leaves the message content, or message body, as flat ASCII text. This document redefines the format of message bodies to allow multi−part textual and non−textual message bodies to be represented and exchanged without loss of information. This is based on earlier work documented in RFC 934 and RFC 1049, but extends and revises that work. Because RFC 822 said so little about message bodies, this document is largely orthogonal to (rather than a revision of) RFC 822. In particular, this document is designed to provide facilities to include multiple objects in a single message, to represent body text in character sets other than US−ASCII, to represent formatted multi−font text messages, to represent non−textual material such as images and audio fragments, and generally to facilitate later extensions defining new types of Internet mail for use by cooperating mail agents.

The mail system supports MIME in several ways:

- When reading a mail message that uses MIME capabilities beyond simple text a MIME interpreter known as `metamail` will be invoked automatically. See metamail(1) for more information.
- You can create mail messages which contain enhanced text, or have multiple parts using `mailx` [see mailx(1)] or `mailto` [see mailto(1)].
- Support for additional message or attachment types, such as audio or visual attachments, can be administered through the file `/etc/mail/mailcap` [see mailcap(4)].

NOTE: The desktop mail utility, `dtmail`, provides these capabilities.

The Administrator's Role

As an administrator, you will only have to become involved with MIME if you want to add support for a new mail attachment format. To add support for a new attachment format, you will need to know several things:

- the command to create this file type,
- the command to display this file type,

- how to describe this file type, and
- access to a terminal when showing a file of this type.

The mailcap File

Support for MIME on your system is facilitated using the `mailcap` file. This is the capabilities file which specifies how to display nontext message formats at the local site. See the mailcap(4) manual page for a description of this file.

Example: MIME Format

Consider the attachment format known as `message/external-body`. This special type of attachment actually refers to a file found on some other system or across an Internet link. To create one, you would use the command `extcompose` with a single argument, which specifies where to put the output. This command generates the output as a series of headers followed by a body which gives additional information needed by some external access methods. This is specified in the `mailcap` file by an entry which says

```
composetyped = extcompose %s;
```

The keyword `composetyped` indicates that headers are output as well as a body. If there were no headers in the output, the keyword `compose` would be used instead. The `%s` indicates where to put the output and will be filled in with a filename.

To display a message/external–body, you would invoke the `showexternal` program, passing in various parameters found on the `Content-type:` header of such a message. This is the first field of the `mailcap` entry for message/external–body and would be specified by

```
showexternal %s %{access-type} %{name} %{site} %{directory} \
    %{mode} %{server};
```

This program interacts with the user, so a terminal is needed. This requires that the `needsterminal` keyword be specified.

We can call this type of attachment ‘‘Reference to data in an external location.’’ This would be specified as part of a `label` field.

Putting this together produces a `mailcap` entry which looks like this:

```
message/external-body; showexternal %s %{access-type} %{name} \
    %{site} %{directory} %{mode} %{server}; \
    needsterminal; composetyped = extcompose %s; \
    description="Reference to data in an external location"
```

As you can see, each field is separated by a semicolon, and continuation lines are specified by putting a backslash at the end of each line.

Locales and Character Sets

MIME requires access to a character set name to describe the character set used by the different locales available on the system. For example, the locale `german` would typically use the character set `iso8859-1`.

When showing a message created with the character set `iso8859-1`, the mail program will check the character set against your current character set and, if they don’t match, lodge a complaint.

The character set in use by a user can be specified by setting the environment variable `MM_CHARSET` to the correct value. However, this requires that the user be informed about yet another environment variable (in addition to the `LOCALE` environment variable).

Eventually, the system will incorporate character set information in the locale–specific system files. Until that time, however, character set information may be made available through a file named `/etc/mail/charset/`*`locale`*, which may be created by the administrator for each locale in use on the system.

These files each have a single line that specifies the character set for that locale. For example, the file `/etc/mail/charset/german` would contain the single line which says `iso8859–1`.