D *The sendmail Program*

From a user's perspective, sending a mail message is a simple procedure. Actually getting the message to its destination can be, as you might know or suspect, fairly complicated. An integral part of the mail system is the **sendmail** program, a message routing facility. It gathers mail from user-level mail sending programs, such as NeXT's Mail.app, and distributes it to mail forwarding or delivery facilities, such as UUCP.

In order to understand **sendmail**, you need to understand **sendmail** configuration files and how they relate to the environment in which they are interpreted (possibly including NetInfo, NIS, and DNS). This appendix presents some background information about the operation of **sendmail**, discusses the configuration files, and gives some debugging examples. The **sendmail** facility is notoriously complex. This appendix covers the basics, and isn't intended as a complete description. See Appendix H, ªSuggested Reading,º for a list of references covering **sendmail**.

Note: The information in this appendix, particularly the tables, is adapted from ªSendmail Installation and Operation Guideº by Eric Allman. This paper is contained in *UNIX System Manager's Manual, 4.3 Berkeley Software Distribution* (see Appendix H).

Warning: The **sendmail** system is designed to handle general electronic mail addressing and forwarding, which is extremely complicated. Don't undertake modifications to the existing configuration files unless you're absolutely sure it's necessary and you're confident you know what you're doing. Be sure to make backup copies of the existing files in case something goes wrong.

Background

The **sendmail** program does four things:

- Determines how to dispose of a messageÐhow to forward it to a mail sending program.
- Rewrites parts of the message header so addresses have the correct format and content.
- Performs alias resolution (converts aliases into user names) and processes **.forward** files for local delivery.
- Forwards mail using SMTP (Simple Mail Transfer Protocol, the TCP/IP network protocol for mail transfer).

Notice that **sendmail** does not by itself deliver mail, and, with the exception of mail forwarded over SMTP, it does not forward mail. Instead, **sendmail** depends on other programs (**/bin/mail** or **uux**, for example) to deliver and forward mail messages.

The **sendmail** program functions as a traffic director. It takes messages from programs which send mail, such as NeXT's Mail.app, determines how to forward the message, and passes the message along to a forwarding or delivery agent. The forwarding agent might be another computer's **sendmail** using SMTP, or another computer's **rmail** program using UUCP. The delivery agent might be **/bin/mail** for local delivery. The instructions that specify how to determine what to do with a given message are in the **sendmail** configuration file.

Configuration Files

In addition to the instructions regarding what to do with a particular mail message, the configuration file also contains information unique to the network running **sendmail**. Each computer can have its own configuration file, but the configuration files on most of the computers in a given network are usually identical. Typically, the

configuration file on the mail server is different from the files on the mail clients, but the files on the clients are the same.

By default, the **sendmail** configuration file is **/etc/sendmail/sendmail.cf**. This might be a symbolic link to another file in **/etc/sendmail**, such as **sendmail.mailhost.cf**. In addition, the location of the configuration file can be stored in the NetInfo database as the value of the **sendmail.cf** property in the **/locations/sendmail** NetInfo directory. For more information, see Chapter 6, ªManaging Electronic Mail.º

Note: Entries in the NetInfo **/locations** directory override **/etc/sendmail/sendmail.cf**.

Note: Some implementations of **sendmail** support a ªfrozenº configuration file, **sendmail.cf**. This file is not supported on NeXT computers, and the ability to create a frozen configuration file has been removed.

Headers and Envelopes

Mail messages contain addresses in two places: the *header* of the message (the **From:**, **To:**, and **Cc:** lines, for example) and the *envelope* of the message, which includes the return address and the current destination addresses (for this copy of the message). Messages are delivered to the destination addresses contained in the envelope. The envelope addresses are those specified on the command line. When you use NeXT's Mail.app, these are hidden. See ªTricks of the Tradeº later in this chapter.

Not all recipients are always shown in the headers. For exam[ple, if you send a ªblindº copy to somone, their address won't be lited anywhere in the received message. Recipients won't see all of the envelope addresses (that's the whole point of a ªblindº copy). The envelope return address can be found in the **From** line (the ªFrom<space>º line as opposed to **From:** line) and **Return-Path:** header lines of a delivered message.

Note: The ªfrom<space>º lines aren't header lines, but rather they separate messages in UNIX mailboxes.

Configuration File Components

The **sendmail** configuration file consists of five basic components. In increasing order of importance and complexity, they are: options, macros, classes, maps, mailers, and rulesets and rewriting rules.

Warning: Modifying your **sendmail** configuration files should be a last resort. First, try to find an existing configuration file (or set of files) known to work for your situation. If you do modify the **sendmail** configuration file, be sure to have a backup copy in case your modifications don't do quite what you intend. It's *extremely* easy to render your mail system inoperative with an inappropriate modification to a **sendmail** configuration file.

Options

Options control various simple aspects of **sendmail** operation. Options can be set both in the configuration file and on the command line used to invoke **sendmail** (usually executed from **/etc/rc**).

In the configuration file, options are set using the **O** directive. Here's an example:

```
OT3d
```

This option says to return mail if it hasn't been delivered within 3 days. Other options control such things as the permissions (ªmodeº) of temporary files, the locations of the aliases file and the spool directory, and the SMTP timeout. For more information, see the UNIX manual page for **sendmail**.

Macros

Macros are simple variables. Some macros are defined by **sendmail** itself, such as the current date and the

sender's host name. Others can be defined in the **sendmail** configuration file. Such macros might include the name of a computer to which mail should be sent if it can't be delivered locally. Macro names consist of a single letter.

A number of built-in macros are available. All have lowercase names. By convention, macros with uppercase names are reserved for user definition.

Macros are defined with the **D** directive:

```
DMetherl
```

This example defines the **M** macro to contain the string **etherl**.

Classes

Classes define collections of equivalent strings (or names, or tokens). This is typically used to indicate that a group of hosts or host names should be treated the same way. For example, all the computers that are UUCP partners might be in a class, as might all the different names for the current computer.

Classes are defined with the **C** and **F** directives. With **C**, you explicitly list the members of the class; with **F**, the members are obtained from a file or from the standard output of a command. Here's an example of each:

```
CRmailhost mail-relay mail-server
FV|/usr/bin/uuname
FA/etc/sysadmins
```

The first line defines the **R** class to contain the strings **mailhost**, **mail-relay**, and **mail-server**. The second defines the **V** class to hold the output of the command **/usr/bin/uuname**. The final example defines the **A** class to hold the contents of the file **/etc/sysadmins**.

Maps

A *map* can be used to look up a token or sequence of tokens in a database, and translate that token or those tokens into something else. For example, a map can be used to translate a user's login name (for example, **tracy**) to some canonical alias (for example, **Tracy_Roykirk**). Maps are used with the **$(¼$)** metasymbol, described below.

Three types of maps are supported: NetInfo, NIS, and dbm. To define a map, you use the K directive, whose syntax is:

```
Kmapname    maptype    mapname    [arguments]
```

Here, *mapname* is a single-letter name for this map, and *maptype* is either **netinfo**, **nis**, or **dbm**. The syntax of both *mapname* and the optional *arguments* depend on the *maptype*, as follows:

Map Type	**Map Name**	**Arguments**
netinfo	[domain**:**]directory	propertykey
nis	map name	
dbm	dbm file (without extension)	

Here are examples of each definition type, NetInfo, NIS, and dbm, respectively. In each case, the Z map is defined; in each case, the intent is to use this map to translate a login name to a canonical alias (known as ªinverse alias translationº).

```
KZnetinfo   localhost/network:/users   email
KZnis       mail.byaddr
KZdbm       /etc/sendmail/inverse_aliases
```

Let's examine the arguments for the first map definition more carefully. The mapname specification, **localhost/network:/users**, has two portions: the domain name (**localhost/network**), and the directory (**/users**). This particular domain name uses tagged domain specification, for efficiency. The assumption here is that this

configuration file's **sendmail** process will be running on a machine which also runs a NetInfo server for a domain tagged **network**. There's also an assumption that we're not willing to look for another server of the domain if, for some reason, the **local netinfod** network is slow to respond. More on this in a moment.

If no domain portion is specified, a standard search up the domain hierarchy will be performed, starting with the local domain. If full domain path notation is usedÐfor example, **/** or **/sales**Ða connection will be made to an available server for the domain. (Note that both no domain specification and a full domain path specification might be expensive in a large domain hierarchy: the domain hierarchy will have to be traversed for each invocation message sent.) A relative domain path can also be used (for example, **.** or **..** or even **../..**); this has the advantage of being able to contact servers on other computers, but still being relatively efficient.

The directory portion specifies the parent of the directory to be accessed, or the directory itself. For example, if the directory portion is **/users** and the token to be translated is **tracy**, then the NetInfo directory **/users/tracy** will be accessed. The token can be included as other than the last component of the directory path, by using the notation **%s**. For example, if the directory portion is **/users/%s/info** and **tracy** is to be translated, then **/users/tracy/info** will be examined. (One way to think about this is that if there's no explicit **%s** in the directory portion specification, there's an implicit **/%s** at the end of the specification.)

The PropertyKey denotes the key of the property whose first value should be substituted for the token(s) being translated. In our example, where we're looking up **tracy** in the NetInfo map shown above, the first value of the email property in **/users/tracy** will be substituted.

Using a Map

We've been discussing maps in a rather abstract manner. Let's take a very specific example: actually implementing inverse alias translation for the addresses of people sending messages. You might want to reread this section after completing the rest of this appendix: we'll be using some information explained in later sections.

The first step in implementing inverse alias translation is to define the appropriate information in NetInfo. You could add a new property, **alias**, to each user's record in **/users** in the root domain, for example. Or, you could add such a property to each user's **info** subdirectory. Let's take the latter approach; using NetInfoManager or an

appropriate script with **niutil**, add all the canonical aliases. Also, be sure that you've defined an appropriate alias in the ***/*aliases** directory to handle the forward translation of, for example, **Tracy_Roykirk** into **tracy**.

We have a few choices regarding which messages are subject to inverse alias translation. It could be just messages being delivered locally, or just messages going out over the Internet, or just messages being forwarded using UUCP. Or, we could perform inverse alias translation on all messages. Let's take this latter option.

As described in ªRuleset Flow and Application,º all sender addresses are translated by Ruleset 1, the Sender Preprocessing ruleset. This is a good place to put inverse alias translation. In the configuration files shipped in NEXTSTEP, Ruleset 1 is empty.

We're going to make two more assumptions here. First, the mailhost machineÐthe machine (or machines) providing centralized mail servicesÐrun a server for the root NetInfo domain, which is tagged **Rhino**. Second, if the NetInfo server for the root domain running on the mailhost doesn't respond, we're willing not to look for another server for the domain.

Let's define the map, and add a rule to Ruleset 1. A good place to define the map is just above Ruleset 1; you only need to modify the configuration file for the mailhost. Here's the map definition, along with the new Ruleset 1.

```
KZnetinfo    localhost/Rhino:/users/%s/info     alias

#  Sender Field Pre-rewriting
S1
R$-          $:$>3$(Z$1$)         invert account to alias
```

The **Z** map is of type **netinfo**, and will query the **netinfod Rhino** running on the local computer. The first value of the **alias** property in the **info** subdirectory of the appropriate user directory will be substituted for the word to be translated.

In Ruleset 1, if there's exactly one token in the sender's address, that token will be translated using the **Z** map. The resulting adddress will be sent back through Ruleset 3 for canicalization. Thus, a sender address of **tracy**

will result in the value of the **alias** property in **/users/tracy/info**Ðfor example, **Tracy_Roykirk**Ðbeing substituted for **tracy**.

Debugging NetInfo Lookups

If you invoke **sendmail** in address test mode (see ªUsing sendmail in Address Test Modeº), you can see the results of the NetInfo lookups. If you want to get some specific, detailed information, you can run **sendmail** in debug mode by specifying the **-d** flag, and giving it the argument **60** (for example, **-d60**). See the section ªAddress Test on a Mail Serverº for an example.

Mailers

A mailer defines the program and options used to forward or deliver mail. Different mailers are used for reasons such as the following:

- Forwarding over different media or using different protocols (UUCP versus SMTP, for example)
- Using different options (such as maximum message length) for different circumstances
- Using different sorts of rewriting for the sender and recipient addresses

Mailers can be named anything, except for three names which have special meaning to **sendmail**: **local**, **prog**, and **error**.

Mailers are defined with the **M** directive:

```
Muucp, P=/usr/bin/uux, F=msDFMhuU, S=13, R=23, E=\n
    A=uux - -r $h!rmail ($u)
```

This example defines the **uucp** mailer. Here's a breakdown of the components:

- **P** (path)ÐDefines the program to be invoked, in this case, **/usr/bin/uux**.
- **F** (flags)ÐDefines the *mailer flags*, which control details of the execution of and arguments to the program invoked for this mailer. A complete description can be found in Eric Allman's paper.
- **S** (sender)ÐSpecifies the ruleset used for rewriting sender addresses of messages sent to this mailer. See ªRuleset Flow and Application,º later in this chapter, for further information.
- **R** (recipient)ÐSame as **S**, but for recipient addresses.
- **E** (eol)ÐSpecifies the end-of-line string, in this case, **\n** (a newline character).
- **A** (argv)ÐDefines the argument vector sent to the program. It begins with a restatement of the program (often in shortened form), and continues with the arguments you would provide to the program on the command line.

Not used in this example is the **M** (maxsize) option, which defines the maximum message length (in bytes). If this maximum size is exceeded, the message is not delivered and an error message is returned to the sender.

The local Mailer

The mailer called **local** must be defined in the configuration file. The **local** mailer is used for final delivery of messages destined for users on the local computer. Alias resolution will only be performed for mail that will be processed by the **local** mailer.

Typically, the **local** mailer will invoke **/bin/mail**, which actually writes a copy of the message to the user's mail spool file:

```
Mlocal, P=/bin/mail, F=rlsDFMmnu; see below for a discussion of sender domain additionP, S=10,
R=20, A=mail -d $u
```

The prog Mailer

The mailer called **prog** also must be defined in the configuration file. This mailer is used for final message delivery when the address specifies a program as the recipient. For example, the following mail alias specifies that the program **/usr/local/bin/ambigmail** should receive the incoming message:

```
"|/usr/local/bin/ambigmail"
```

The **prog** mailer is invoked in place of the **local** mailer for such addresses. Typically, the **prog** mailer will invoke **/bin/sh**:

```
Mprog, P=/bin/sh, F=lsDFMeup, S=10, R=20, A=sh -c $u
```

The error Mailer

The **error** mailer is built into **sendmail**, and shouldn't be defined in the configuration file. This mailer is used to report errors, such as an unknown local host, to the sender and possibly to **postmaster**, an alias for the users responsible for the mail installation. For an example using the **error** mailer, see the next section.

Error messages can also be generated from other sources. For example, if the **local** mailer determines that a user doesn't exist locally, an error will be generated, but the **error** mailer isn't used to report it.

Rulesets and Rewriting Rules

Much of a **sendmail.cf** file consists of instructions for interpreting and modifying mail addresses. These instructions are called *rewriting rules*, and they're grouped together into *rulesets*. A ruleset contains a collection of rules to perform a certain function. For example, one ruleset transforms addresses into a standard (*canonical*) format which is easier for the other rulesets to parse.

Rulesets are named with a number, and are defined using the **S** directive:

```
S30
```

A ruleset is followed immediately by the rewriting rules that make up that ruleset. Rewriting rules are defined with the **R** directive:

```
R$+@$+    $:$1<@$2>    focus on domain
```

This rule is made up of three components, separated by tab characters: the left-hand side, the right-hand side, and a comment.

Left-hand Side

The pattern contained in the left-hand side of a rule is compared to an address. The pattern is similar to a regular expression, except that it compares *tokens*, or words, rather than characters. For example, the following address has five tokens:

```
amm@Tute.EDU
```

The five tokens are **amm**, @, **Tute**, **.**, and **EDU**.

The following table describes the metasymbols used in left-hand side patterns:

Metasymbol	**Meaning**
$*	Match zero or more tokens.
$+	Match one or more tokens.
$-	Match exactly one token.
$=*c*	Match any token in Class *c*.
$~*c*	Match any token not in Class *c*.

Here's the example rewriting rule again:

```
R$+@$+      $:$1<@$2>      focus on domain
```

The left-hand side of this rule matches one or more tokens (**$+**), followed by the token @, and one or more additional tokens (**$+**).

Right-hand Side

The right-hand side of a rule specifies what to do with an address if the left-hand side of the rule matches it. The following table describes the metasymbols used in the right-hand side of a rewriting rule:

Metasymbol	Meaning
$*n*	Substitute the tokens that match metasymbol number *n* from the left-hand side.
$[*name*$]	Transform host name (or address) *name* into standard form (*canonicalize*).
$>*n*	Call ruleset number *n*.
$#*name*	Resolve the address to the mailer *name*.
$@*name*	Specify the host as *name*.
$:*name*	Specify the user as *name*.
${*m name*$}	Look up *name* in the NIS map specified in the macro *m* [obsolete; superceded by $(¼$)].
$(*m word*$)	Look up *word* in the map specified by *m* (*m* is defined using the **K***m* directive, below).

Note: The $# and the $@ and the $: metasymbols are used together in the form $#mailer $@host $:user.

In addition to these metasymbols, there are two prefixes which can be used to modify the behavior of the rule or ruleset, as described in the following table:

Prefix	Meaning
$:	Terminate this rule immediately, but allow the ruleset to continue.
$@	Return from this ruleset with the remainder of the right-hand side as the result of the ruleset.

Note: The two above prefix usages apply only when they appear first on the Right Hand Side.

Here's the example rewriting rule again:

```
R$+@$+      $:$1<@$2>      focus on domain
```

The right-hand side rewrites the address into the following form:

```
Tokens1<@Tokens2>
```

If this rule were applied to the address **amm@Tute.EDU**, *Tokens1* would match **amm**, and *Tokens2* would match **Tute.EDU**. The right-hand side of the rule transforms the address into the following:

```
amm<@Tute.EDU>
```

By putting the domain portion of the address between angle brackets (<>), it's more easily identified by other rules (this is called *focusing*).

The right-hand side prefix **$:** causes the rule to terminate, preventing application of the rule to the rewritten address. (The rewritten address might still match the left-hand side, which would otherwise cause the rule to be applied to the rewritten address.)

Here's an example of a rewriting rule that invokes the **error** mailer:

```
R$*<@$*.LOCAL>$*    $#error $:Never heard of host $2 in domain $m
```

If the left-hand side matches the address, the error mailer is called to return the indicated error message (everything after **$:**). Tab characters can't be used in the text of the error message since a tab separates the right-hand side of a rewriting rule from the comment.

Note: The **error** mailer should only be invoked in Ruleset 0 (see the next section).

Translating Addresses

Perhaps the most important aspect of **sendmail** configuration files is how the rulesets are used to translate a mail address.

Address Syntax

Electronic mail addresses have many different formsÐthat's why **sendmail** configuration files are so complex.

A mail address is usually in one of three forms:

- Simple user name or alias, such as **suser**, **Sandy_User**, or **all_sales**
- User name or alias at some other computer, such as **suser@Rhino.COM**
- User name or alias on another computer reached using UUCP, such as **rhino!suser**

Other address forms are possible, including combining address forms. The following table describes some sample mail addresses:

Address	**Interpretation**
amm	Local user or alias **amm**
amm@rhino	**amm** on the computer **rhino** in the local domain
rhino!amm	**amm** on the computer **rhino** reached using UUCP
amm@rhino.UUCP	**amm** on the computer **rhino** reached using UUCP
rhino!hippo!conure!amm	**amm** on the computer **conure** reached with UUCP through **rhino** and then **hippo**

amm@Rhino.COM	**amm** in the domain **Rhino.COM**
amm@flyer.Rhino.COM	**amm** in the domain **flyer.Rhino.COM**; often, this is the computer **flyer** in the domain **Rhino.COM**
<@Rhino.COM:amm@Tute.EDU>	**amm** in the domain **Tute.EDU**, reached through the domain **Rhino.COM**
amm%rhino.UUCP@Tute.EDU	**amm** on the computer **rhino.UUCP** reached through the domain **Tute.EDU**
amm%Rhino.COM@Tute.EDU	**amm** at **Rhino.COM** reached through **Tute.EDU**

The **sendmail** configuration file translates all addresses into conforming addresses, called *domain-style addresses*. A simple domain-style address is *user@domain*, where *domain* is the name of a computer or collection of computers (a network, for example). If *domain* is a collection of computers, one of those computers will be designated to handle the incoming mail for the collection. The **sendmail** program (or its equivalent) on the computer that handles mail for *domain* interprets *user*.

For example, the address **rhino!suser**, which isn't a domain-style address, will be translated to **suser@rhino.uucp**. In this case, **rhino.uucp** is a notation used for convenience: there isn't really a **uucp** domain, and therefore no subdomain called **rhino.uucp**. The translation into a standard, or canonical, domain-style address is called *canonicalization*.

In general, mail addresses aren't case-sensitiveÐthe address **amm@rhino** is the same as **amm@Rhino**. The exception to this is UUCP addressesÐ**amm@rhino.uucp** is *not* the same as **amm@Rhino.uucp**.

Ruleset Flow and Application

Rulesets are applied to an address in a specific order. This section describes which rulesets are applied to

different kinds of addresses. The rulesets themselves are described in the next section.

The following figure shows the rulesets used during mailer resolutionÐdetermining what to do with a message being sent to a given address.

F0.eps ,

The envelope address (which might be different from the addresses in the message itself, or might not appear in the message) is processed first by Ruleset 3, and then by Ruleset 0. The function of Ruleset 3 is to convert the address into standard (canonical) form, and to ªfocusº **sendmail** on the domain part of the address by placing it in angle brackets (<>). Ruleset 0 determines what to do with the message sent to that address.

The next figure shows the rulesets used to rewrite sender addresses in the message header, such as the **From:** line.

F1.eps ,

Ruleset 3 is called first, possibly followed by a process called *sender domain addition* (represented here by the box labeled **D**). Sender domain addition adds a domain portion to the address, if necessary; it is explained below. Next, Ruleset 1 is invoked, which performs general sender address preprocessing, such as translating a login name to an alias. Then, the ruleset specified by the **S** option in the mailer definition is invoked, followed by Ruleset 4, which performs general postprocessing, including undoing some of the focusing performed by Ruleset 3.

The next figure shows the rulesets used to rewrite recipient addresses in the message header, such as the **To:** line.

F2.eps ,

Here, Ruleset 3 and D are as before. Ruleset 2 performs general recipient address processing (frequently not

required, so Ruleset 2 is often empty). Then, the ruleset specified by the **R** option in the mailer definition is invoked, followed by Ruleset 4, which performs general postprocessing.

The following figure is a combination of the previous three figures, showing how the various rulesets are used when processing an address. Which path an address takes through the rulesets depends on the type of address. This diagram is adapted from the paper by Eric Allman, cited earlier, and appears in many **sendmail** references.

F3.eps ,

Specific Rulesets

This section describes in detail the various rulesets introduced in the previous section.

Ruleset 3ÐCanonicalize and Focus

Ruleset 3 translates an address into standard form (canonicalization), and encloses the domain portion of the address between angle brackets (focusing). The example rewriting rule used earlier is a sample from Ruleset 3.

Here are a couple examples of how addresses are translated by Ruleset 3:

Original Address	**After Translation**
amm@Rhino.COM	amm<@Rhino.COM>
rhino!amm	amm<@rhino.uucp>

Ruleset 0ÐMailer Resolution

Ruleset 0 determines what to do with the message for an address: which mailer should be used to forward or

deliver the mail, which host receives the message if it's to be forwarded, and which address should be given to the forwarding or delivery agent. This process is called *resolving* the address. The output of Ruleset 0 contains three things:

- Mailer to be used to forward or deliver the mail (**ddn**, for example)
- Host to which the message should be sent (**Tute.EDU**, for example)
- Address of the user receiving the mail (**amm@Tute.EDU**, for example)

If the mail is to be delivered locally, the host will be empty.

The next table shows the result of applying Ruleset 0 to a number of addresses. This table assumes that address resolution is occurring on the mail server (the computer responsible for delivering local mail and forwarding remote mail), and that mail destined for off-site delivery is forwarded over the Internet.

Address	**Mailer**	**Host**	**User**
suser	local		suser
suser@rhino	ether	rhino	suser@rhino
suser@Tute.EDU	ddn	Tute.EDU	suser@Tute.EDU
rhino!suser	uucp	rhino	suser
suser@rhino.uucp	uucp	rhino	suser
rhino!conure!suser	uucp	rhino	conure!suser
suser%Rhino.COM@Tute.EDU	ddn	Tute.EDU	suser%Rhino.COM@Tute.EDU

If you want to use some other mailer to forward off-site mail, you need to define that mailer with the **M** directive, as described earlier. Additionally, the **M** macro should be defined as the name of that mailer, the **R** macro as the preferred name of the remote forwarding computer, and the **R** class as the preferred name and any aliases for the remote forwarding computer.

Ruleset DÐSender Domain Addition

There really isn't a Ruleset D; it's actually a built-in process called sender domain addition. The sender's address

is run through the mailer resolution sequence (Ruleset 3, then Ruleset 0), then **sendmail** checks to see if the **C** mailer flag is present for the mailer that the message was *received* from. If the flag is present, the domain portion of the sender's address is added to any addresses in the header that don't already have a domain portion (that is, that appear to be local addresses). This makes sure that any replies to the message are delivered to the correct address. For example, if the message is sent from the user **amm**, sender domain addition might convert the address to **amm@Rhino.COM**.

Ruleset 1ÐStandard Sender Address Preprocessing

Ruleset 1 provides an opportunity to do site-wide processing of a sender's address. For example, a site might implement a standard format for mail addresses, so that, regardless of a user's actual login name, mail from that user would appear to come from this standard address. Specifically, you might want to translate a login name like **rkabir** to **Randy_Kabir**.

Ruleset 2ÐStandard Recipient Address Preprocessing

Just as you can do standard preprocessing of sender addresses with Ruleset 1, you can perform standard preprocessing for recipient addresses with Ruleset 2. For example, you might perform host name translation for some of the computers on your local network. Usually, little if any recipient address preprocessing is required.

Ruleset 4ÐFinal Address Postprocessing

Ruleset 4 undoes some of the address translation performed by Ruleset 3. It also defocuses the address by removing the angle brackets around the domain portion. For example, it might convert **amm<@rhino.uucp>** back into **rhino!amm**.

Information Services

Depending on how the system is configured, **sendmail** uses various sources for network-wide information, including NetInfo, DNS, NIS, and some UNIX flat files. How these are used depends on the information needed.

- User NamesÐWhen **sendmail** looks for a user name, NetInfo is checked first. If the name is not found in NetInfo, and if NIS is enabled, the appropriate UNIX file will be examined, and then the NIS map will be checked.
- Host NamesÐIn addition to NetInfo and NIS, the DNS is searched for host names. Once a message's destination has been determined, the DNS is searched to determine if a *mail exchanger* exists for the destination computer. A mail exchanger is a computer that handles all the mail destined for another computer. Mail exchanger records in the DNS are called *MX records*.

  If an MX record is found for the destination domain, that computer is used for the destination in the mail transmission (but *not* in the message headers). The information in the MX record includes the address of the mail exchanger, so no further reference to NetInfo or NIS is made to resolve the address of the mail exchanger.

  The order of search is NetInfo, DNS, NIS, then DNS for MX records.
- AliasesÐAliases are translated only for messages being delivered with the **local** mailer. To resolve an alias, **sendmail** looks for the address in three places, in this order: the **/aliases** NetInfo directory, the **mail.aliases** NIS map (if NIS is running), and the file specified by the **A** option in the **sendmail** configuration file (usually **/etc/sendmail/aliases**). The latter locations are only checked if the address isn't found in an earlier one. If the address is not found in any of these places, it's left as is. Note that this is independent of the **I** mailer flag.

After alias resolution, **sendmail** looks for a file called **.forward** in the recipient's home directory. This file, if present, specifies that the mail should be forwarded: to another user (not necessarily local), a file, or a program.

The following table describes examples of each of the variations of the **.forward** file.

Type	Format	Example
User	Mail address	amm@Tute.EDU
Literal user	Escaped address	\amm
File	Full path	/Net/rhino/Users/amm/.maillog
Program	| *program*	"|/usr/local/bin/answermail -a"

Important: If a file is specified, it must already exist in order for mail to be written to itÐ**sendmail** will not create a new file.

If a command is specified in a **.forward** file, the command is run in the mail queue spool directory as if executed by the mail recipient. The mail queue spool directory is specified by the **Q** option (usually **/usr/spool/mqueue**). If the command is a shell script, the user's **.profile** or **.cshrc** is run. If arguments are provided to the command, the entire command line must be enclosed in quotes, as in the previous example.

To forward the mail to multiple recipients (for example, to a file and to a program), specify each entry on a separate line in the **.forward** file.

Warning: In order to resolve forwarding, **sendmail** attempts to access the recipient's home directory. If the home directory is on an imported file system and the file server is unavailable, the **sendmail** process might have to wait until the server becomes available.

Debugging

Debugging **sendmail** problems can be a complex, frustrating experience. The suggestions in this section are a starting point.

The mail and sendmail -v Option

The **-v** (verbose) option to the Berkeley **mail** command displays the **sendmail** conversation or actions as the mail is sent. What follows is an example of this command. The example shows user input in boldface and comments in a different typeface.

```
rhino [~]-175% mail -v rkabir
```

Send mail in verbose mode to **rkabir**.

```
Subject: Testing
This is a test
.
```

Enter a message, ending with a period on a line by itself.

```
Cc:
```

No copies to anyone.

```
rkabir... Connecting to mailhost via etherl...
```

Local computer attempts to send mail to **rkabir**. Since this computer is a mail client, mail is sent to the mail server (**mailhost**) using the **etherl** mailer.

```
Trying 192.42.172.66...  connected.
```

192.42.172.66 is the Internet address of the mail server.

```
220 cockatoo.Rhino.COM Sendmail NX5.67c/NX3.0M ready at Wed, 15 Jul 92 09:14:18 -0700
```

The greeting from the mail server (**cockatoo**).

```
>>> HELO conure.Rhino.COM
250 cockatoo.Rhino.COM Hello conure.rhino.com, pleased to meet you
```

Both computers identify themselves to each other.

```
>>> MAIL From:<amm>
250 <amm>... Sender ok
```

Mail is being sent from the user **amm**; the sender address is accepted.

```
>>> RCPT To:<rkabir>
250 <rkabir>... Recipient ok
```

Mail is being sent to **rkabir**; the recipient address is accepted. If there were some problem with the address, an error message would be printed.

```
>>> DATA
354 Enter mail, end with "." on a line by itself
```

```
>>> .
250 Mail accepted
```

Mail message sent and received.

```
>>> QUIT
221 cockatoo.Rhino.COM delivering mail
rkabir... Sent
```

You can also invoke **sendmail** directly with the verbose option. The output is identical to that from a verbose **mail** session. When you run **sendmail** directly, you must supply all the headers that your mail sending program provides for you (usually including **From:**, **To:**, **Cc:**, and **Subject:**). Delivery will be attempted to the addresses provided on the command line regardless of the headers.

Using sendmail in Address Test Mode

You can run **sendmail** in *address test mode*, which displays the results of processing an address by the various rulesets in the configuration file. In address test mode, you can specify the rulesets to be applied to a specific address. You specify a ruleset by number, with a sequence of rulesets separated by commas and the rulesets separated from the address by a space. To see how a specific address would be processed by **sendmail**, specify the sequence of rulesets according to the information in ªRuleset Flow and Applicationº earlier in this chapter.

Normally, the address is resolved to a mailer first. Run the envelope recipient address (the address of the actual recipient) through Ruleset 0 to resolve it to a mailer. The envelope address might be different from the address in the header. For example, the address of a recipient of a blind copy of a message will not show up in the message header, but will be on the envelope. However, a recipient's header address and envelope address will usually be the same.

After invoking Ruleset 0 on an address, check the definition of the resulting mailer to determine which sequence of rulesets to use for the sender address, and which to use for the recipient address. Remember, the **S** and **R** macros in a mailer definition specify the rulesets for sender and recipient addresses.

Note: When mail is being delivered, **sendmail** normally performs sender domain addition. In address test mode, however, **sendmail** doesn't perform sender domain addition. Remember to add the sender domain yourself to any addresses that apply when running **sendmail** in address test mode. See ªRuleset DÐSender Domain Additionº earlier in this chapter for details.

What follows is a series of annotated sessions with **sendmail** in address test mode, both on a mail client and a mail server. The examples show user input in boldface and comments in a different typeface.

Address Test on a Mail Client

The first session examines **sendmail** ruleset processing on a mail client. This example assumes you're sending mail from **cockatoo**, in the domain **Rhino.COM**. Other computers on the local network include the mail server (**mailbox**, with the alias **mailhost**) and the computer **conure**.

Remember that mail clients know nothing about mail delivery. Instead, they depend on the mail server for forwarding services.

Here's the definition of the **etherl** mailer from the file **sendmail.sharedsubsidiary.cf**:

```
Metherl, P=[TCP], F=msDFuXn, S=12, R=22, A=TCP $h
```

Local Address

```
cockatoo [~]-329% /usr/lib/sendmail -bt
```

Run **sendmail** in address test mode.

```
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No automatic ruleset 3 call]
```

A greeting is printed. Unlike some other sendmails, we don't automatically run Ruleset 3; this allows you to test individual rulesets without interference from Ruleset 3.

```
> 3,0 amm
```

Run Rulesets 3 and 0 on **amm** to resolve the address to a mailer.

```
rewrite: ruleset 3 input: "amm"
rewrite: ruleset 3 returns: "amm"
rewrite: ruleset 0 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 0 returns: $# "etherl" $@ "mailhost" $: "amm"
```

Ruleset 3 is invoked, followed by Ruleset 0, according to the instructions. Ruleset 0, in turn, calls Ruleset 9 as a subroutine. The input and output of each ruleset are shown. Ruleset 0 returns the values **etherl** (mailer), **mailhost** (host to which the message should be sent), and **amm** (the user to receive the mail). The strings **$#**, **$@**, and **$:** indicate that the words following are the mailer, the host, and the user.

```
> 3,1,12,4 tracy
```

Run the sender's address (**tracy**) through the appropriate ruleset sequence. The **etherl** mailer definition specifies sender Ruleset 12 (S=12).

```
rewrite: ruleset 3 input: "tracy"
rewrite: ruleset 3 returns: "tracy"
```

Execution of Ruleset 3.

```
rewrite: ruleset 1 input: "tracy"
rewrite: ruleset 1 returns: "tracy"
```

Ruleset 1 doesn't modify the address.

```
rewrite: ruleset 12 input: "tracy"
rewrite: ruleset 12 returns: "tracy"
```

Neither does Ruleset 12, the mailer-specific ruleset.

```
rewrite: ruleset 4 input: "tracy"
rewrite: ruleset 9 input: "tracy"
rewrite: ruleset 9 returns: "tracy"
```

Ruleset 4 calls Ruleset 9.

```
rewrite: ruleset 4 returns: "tracy"
```

No changes made to the address.

```
> 3,2,22,4 amm
```

Run the recipient address (**amm**) through the appropriate ruleset sequence. The **etherl** mailer definition specifies recipient Ruleset 22 (R=22).

```
rewrite: ruleset 3 input: "amm"
```

```
rewrite: ruleset 3 returns: "amm"
rewrite: ruleset 2 input: "amm"
rewrite: ruleset 2 returns: "amm"
rewrite: ruleset 22 input: "amm"
rewrite: ruleset 22 returns: "amm"
rewrite: ruleset 4 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 4 returns: "amm"
```

As with the sender's address, no changes were needed.

Simple Remote Address

Next, mail to be sent to a remote address is examined (**amm** at the site **Tute.EDU**). This is still a mail client, so the **sendmail** configuration file is the same, including the definition of the **etherl** mailer.

```
> 3,0 amm@Tute.EDU
```

Run the recipient address through Rulesets 3 and 0 to resolve to a mailer.

```
rewrite: ruleset 3 input: "amm" "@" "Tute" "." "EDU"
rewrite: ruleset 6 input: "amm" "<" "@" "Tute" "." "EDU" ">"
```

Ruleset 3 accomplishes focusing before calling Ruleset 6.

```
rewrite: ruleset 6 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 0 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 0 returns: $# "etherl" $@ "mailhost" $: "amm" "<" "@" "tute" "." "edu" ">"
```

The address is resolved to the **etherl** mailer. The message would be sent to **mailhost**, to be sent along to **amm@tute.edu**.

```
> 3,1,12,4 tracy
```

Run the sender address through the appropriate ruleset sequence. Again, Ruleset 12 is defined as the mailer-specific sender ruleset (S=12).

```
rewrite: ruleset 3 input: "tracy"
```

```
rewrite: ruleset 3 returns: "tracy"
rewrite: ruleset 1 input: "tracy"
rewrite: ruleset 1 returns: "tracy"
rewrite: ruleset 12 input: "tracy"
rewrite: ruleset 12 returns: "tracy"
rewrite: ruleset 4 input: "tracy"
rewrite: ruleset 9 input: "tracy"
rewrite: ruleset 9 returns: "tracy"
rewrite: ruleset 4 returns: "tracy"
```

No changesÐmail is forwarded to the mail server, where changes might be made.

> **3,2,22,4 amm@tute.edu**

Send the recipient address through the ruleset sequence. Ruleset 22 is specified in the **etherl** mailer definition (R=22).

```
rewrite: ruleset 3 input: "amm" "@" "tute" "." "edu"
rewrite: ruleset 6 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 2 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 2 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 22 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 22 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 4 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 4 returns: "amm" "@" "tute" "." "edu"
```

Again, the address is unchanged.

Notice that mail sent to the mail server will never have its header addresses rewritten.

UUCP Address with Local Connection

Now examine the transformations that occur to a message address when sent over UUCP, from **amm** to **ranger! sandy** This example assumes the computer **ranger** is a UUCP partner of **cockatoo**. Here's the **uucp** mailer

definition from **sendmail.sharedsubsidiary.cf**:

```
Muucp,    P=/usr/bin/uux, F=msDFMhuU, S=13, R=23, E=\n,
          A=uux - -r $h!rmail ($u)
```

> **3,0 ranger!sandy**

Resolve the recipient address to a mailer.

```
rewrite: ruleset 3 input: "ranger" "!" "sandy"
rewrite: ruleset 6 input: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 6 returns: "sandy" "<" "@" "ranger" "." "uucp" ">"
```

Canonicalization and focusing accomplished.

```
rewrite: ruleset 3 returns: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 0 input: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 0 returns: $# "uucp" $@ "ranger" $: "sandy"
```

This address is resolved to the **uucp** mailer, rather than being forwarded to **mailhost**. The assumption is that even mail clients can have their own UUCP partners.

> **3,1,13,4 amm**

Run the sender adress through the appropriate ruleset sequence; the **uucp** mailer definition specifies Ruleset 13 for sender addresses (S=13).

```
rewrite: ruleset 3 input: "amm"
rewrite: ruleset 3 returns: "amm"
```

Ruleset 3 canonicalization and focusing (no changes needed here).

```
rewrite: ruleset 1 input: "amm"
rewrite: ruleset 1 returns: "amm"
```

No changes by Ruleset 1.

```
rewrite: ruleset 13 input: "amm"
rewrite: ruleset 5 input: "amm"
```

Ruleset 13 calls Ruleset 5, which does UUCP address conversion.

```
rewrite: ruleset 5 returns: "amm"
rewrite: ruleset 13 returns: "cockatoo" "!" "amm"
```

Ruleset 13 added **cockatoo!** to the front of the address, to indicate that the sender's address, relative to the receiving computer, is **cockatoo!amm**.

```
rewrite: ruleset 4 input: "cockatoo" "!" "amm"
rewrite: ruleset 9 input: "cockatoo" "!" "amm"
```

```
rewrite: ruleset 9 returns: "cockatoo" "!" "amm"
rewrite: ruleset 4 returns: "cockatoo" "!" "amm"
```

Since the address started off as a local address, no focusing was necessary. So there's nothing to do in Ruleset 4 for unfocusing. In addition, the address is already in canonical form.

```
> 3,2,23,4 ranger!sandy
```

Translate the recipient's address; the **uucp** mailer definition includes R=23.

```
rewrite: ruleset 3 input: "ranger" "!" "sandy"
rewrite: ruleset 6 input: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 6 returns: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 3 returns: "sandy" "<" "@" "ranger" "." "uucp" ">"
```

A canonical, focused address is returned.

```
rewrite: ruleset 2 input: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 2 returns: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 23 input: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 5 input: "sandy" "<" "@" "ranger" "." "uucp" ">"
rewrite: ruleset 5 returns: "ranger" "!" "sandy"
rewrite: ruleset 23 returns: "ranger" "!" "sandy"
rewrite: ruleset 4 input: "ranger" "!" "sandy"
rewrite: ruleset 9 input: "ranger" "!" "sandy"
rewrite: ruleset 9 returns: "ranger" "!" "sandy"
rewrite: ruleset 4 returns: "ranger" "!" "sandy"
```

Nothing else to do with this address.

Remote Address on the Local Network

As a further example, examine what happens when mail is sent directly to a computer other than the mail server on the local network. Here's the definition of the **etherl** mailer from the file **sendmail.sharedsubsidiary.cf** again:

```
Metherl, P=[TCP], F=msDFuXn, S=12, R=22, A=TCP $h

> 3,0 amm@cockatoo
rewrite: ruleset 3 input: "amm" "@" "cockatoo"
rewrite: ruleset 6 input: "amm" "<" "@" "cockatoo" ">"
```

Focus on the host part.

```
rewrite: ruleset 6 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
```

Canonicalize: indicate that the address is local.

```
rewrite: ruleset 3 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 0 input: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 30 input: "amm"
```

Ruleset 30 simply invokes Ruleset 3 and then Ruleset 0. There is redundant routing information in this local address; this is handled in Ruleset 0. Since you're on a network with a shared mail server, the assumption is that all mail destined for local addresses goes to the mail server. Any other routing on the local network is superfluous.

```
rewrite: ruleset 3 input: "amm"
```

Restart the address resolution, now with a local address.

```
rewrite: ruleset 3 returns: "amm"
rewrite: ruleset 0 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 0 returns: $# "etherl" $@ "mailhost" $: "amm"
rewrite: ruleset 30 returns: $# "etherl" $@ "mailhost" $: "amm"
rewrite: ruleset 0 returns: $# "etherl" $@ "mailhost" $: "amm"
```

Resolved to the **etherl** mailer.

If you run the header addresses through the appropriate rulesets, you find that no changes are made, just as with the other examples where the address was resolved to use the **etherl** mailer.

Here's what happens if you address mail to a user specifically at the mail server:

```
> 3,0 amm@mailhost
rewrite: ruleset 3 input: "amm" "@" "mailhost"
rewrite: ruleset 6 input: "amm" "<" "@" "mailhost" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "LOCAL" ">"
```

Ruleset 6 sees that **mailhost** is an alias for the local mail server, and indicates that the address is a local one.

```
rewrite: ruleset 3 returns: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 0 input: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 9 input: "amm" "<" "@" "rhino" "." "com" ">"
```

Ruleset 0 invokes Ruleset 9 after having converted **LOCAL** to **rhino.com**, this computer's domain.

```
rewrite: ruleset 9 returns: "amm" "<" "@" "rhino" "." "com" ">"
rewrite: ruleset 0 returns: $# "etherl" $@ "mailhost" $: "amm" "<" "@" "rhino" "." "com" ">"
```

As expected, the mail goes to the mail server. Notice, though, that the envelope address will be **amm@rhino.com**. You might assume that the mail server will recognize an address like this as a local address.

If you examine the header addresses, you'll see that the sender address is unchanged, but the recipient address is modified:

```
> 3,2,22,4 amm@mailhost
```

Run the recipient address through the ruleset sequence.

```
rewrite: ruleset 3 input: "amm" "@" "mailhost"
rewrite: ruleset 6 input: "amm" "<" "@" "mailhost" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "LOCAL" ">"
```

The address is now focused and canonicalized.

```
rewrite: ruleset 2 input: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 2 returns: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 22 input: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 22 returns: "amm"
```

Ruleset 22, the mailer-specific ruleset, does something in this case. It converts **amm@LOCAL** to **amm**, thus making the header address appear to be local.

```
rewrite: ruleset 4 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 4 returns: "amm"
```

Address Test on a Mail Server

Now that you've seen what happens to messages and addresses on a mail client, here's what happens with some similar addresses on a mail server. The examples assume that the server is connected to the Internet, and is both the local mail server and the Internet mail gateway. Again, the name of the mail client is **cockatoo**; the name of the mail server is **mailbox**.

Here's the **local** mailer definition from **sendmail.mailhost.cf**:

```
Mlocal,    P=/bin/mail, F=rlsDFMmnP, S=10, R=20, A=mail -d $u
```

Local Address

```
mailbox [~]-20% /usr/lib/sendmail -bt
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No automatic ruleset 3 call]
> 3,0 amm
rewrite: ruleset 3 input: "amm"
rewrite: ruleset 3 returns: "amm"
rewrite: ruleset 0 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 0 returns: $# "local" $: "amm"
```

The address is resolved to the **local** mailer.

```
> 3,1,10,4 tracy
```

The local mailer defines its sender ruleset (S=) as 10.

```
rewrite: ruleset 3 input: "tracy"
rewrite: ruleset 3 returns: "tracy"
rewrite: ruleset 1 input: "tracy"
rewrite: ruleset 1 returns: "tracy"
rewrite: ruleset 10 input: "tracy"
rewrite: ruleset 10 returns: "tracy"
rewrite: ruleset 4 input: "tracy"
rewrite: ruleset 9 input: "tracy"
rewrite: ruleset 9 returns: "tracy"
rewrite: ruleset 4 returns: "tracy"
```

```
> 3,2,20,4 amm
```

The local mailer defines its recipient ruleset (R=) as 20.

```
rewrite: ruleset 3 input: "amm"
rewrite: ruleset 3 returns: "amm"
rewrite: ruleset 2 input: "amm"
rewrite: ruleset 2 returns: "amm"
rewrite: ruleset 20 input: "amm"
rewrite: ruleset 20 returns: "amm"
rewrite: ruleset 4 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 4 returns: "amm"
```

Not surprisingly, no changes were made to either the sender or the recipient addresses.

Debugging Inverse Alias Translation

Let's take a look at the output from running address test mode with NetInfo lookup debugging turned on.

```
cockatoo [~]-330% /usr/lib/sendmail -bt -d60 -Cinvert.cf
```

Invoke sendmail in address test mode with NetInfo debugging output. Use the configuration file **invert.cf**.

```
Version NX5.67e
defining map Z: netinfo localhost/Rhino:/users/%s/info alias
```

Define a NetInfo map called **Z**, accessing **netinfod Rhino** on the local computer. Use the value of the **alias** property in the **info** subdirectory of the sender's **/user** directory.

```
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No automatic ruleset 3 call]
> 3,1 tracy
```

Let's assume the user **tracy** is sending this message

```
rewrite: ruleset  3   input: "tracy"
rewrite: ruleset  3 returns: "tracy"
rewrite: ruleset  1   input: "tracy"
```

Ruleset 1 is where inverse alias translation is implemented.

```
looking up map Z: tracy (NULL)
```

Accessing the **Z** map, to translate **tracy**.

```
lookup result: Tracy_Roykirk
```

Tracy_Roykirk is the first value of the **alias** property in **/users/tracy/info**.

```
rewrite: ruleset  3   input: "Tracy_Roykirk"
rewrite: ruleset  3 returns: "Tracy_Roykirk"
rewrite: ruleset  1 returns: "Tracy_Roykirk"
```

The preferred sender's address is **Tracy_Roykirk**.

Simple Remote Address

Here, mail is sent to a remote computer, with a sender address of **amm**, this time at **Tute.EDU**. Here's the **ddn** mailer definition from **sendmail.mailhost.cf**:

```
Mddn,    P=[TCP], F=msDFMuCX, S=22, R=22, A=TCP $h, E=\r\n
```

```
> 3,0 amm@Tute.EDU
rewrite: ruleset 3 input: "amm" "@" "Tute" "." "EDU"
rewrite: ruleset 6 input: "amm" "<" "@" "Tute" "." "EDU" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 0 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 0 returns: $# "ddn" $@ "tute" "." "edu" $: "amm" "<" "@" "tute" "." "edu" ">"
```

The address is resolved to the **ddn** mailer.

```
> 3,1,22,4 tracy
```

The **ddn** mailer defines its sender ruleset as 22.

```
rewrite: ruleset 3 input: "tracy"
rewrite: ruleset 3 returns: "tracy"
rewrite: ruleset 1 input: "tracy"
rewrite: ruleset 1 returns: "tracy"
```

If Ruleset 1 implemented a translation to a standard (canonical) mail address such as **Tracy_Roykirk**, then its output here would be **tracy_roykirk** instead.

```
rewrite: ruleset 22 input: "tracy"
rewrite: ruleset 22 returns: "tracy" "<" "@" "mailbox" "." "rhino" "." "com" ">"
```

Ruleset 22 adds the fully-qualified host name. It also performs the standard (canonical) address translation

just mentioned (see **sendmail.mailhost.cf** for details). Here, the assumption is that the standard translation should be done only for messages being sent to remote computers. You might also choose not to include the host name of the mail server and use only **rhino.com** as your return site. (This would require ensuring that the external network understands that **rhino.com** means **mailbox.rhino.com**.)

```
rewrite: ruleset 4 input: "tracy" "<" "@" "mailbox" "." "rhino" "." "com" ">"
rewrite: ruleset 9 input: "tracy" "<" "@" "mailbox" "." "rhino" "." "com" ">"
rewrite: ruleset 9 returns: "tracy" "<" "@" "mailbox" "." "rhino" "." "com" ">"
rewrite: ruleset 4 returns: "tracy" "@" "mailbox" "." "rhino" "." "com"
```

```
> 3,2,22,4 amm@Tute.EDU
```

The **ddn** mailer defines its recipient ruleset as 22, the same as the sender ruleset.

```
rewrite: ruleset 3 input: "amm" "@" "Tute" "." "EDU"
rewrite: ruleset 6 input: "amm" "<" "@" "Tute" "." "EDU" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 2 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 2 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 22 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 22 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 4 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 input: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 9 returns: "amm" "<" "@" "tute" "." "edu" ">"
rewrite: ruleset 4 returns: "amm" "@" "tute" "." "edu"
```

No change to the recipient's address.

Remote Address on the Local Network

Things aren't quite as straightforward on a mail server, even with simple remote addresses. These examples show sending mail explicitly to the mail server (from the mail server), and sending mail explicitly to another local host. Here's the **ether** mailer definition from **sendmail.mailhost.cf**:

```
Mether,    P=[TCP], F=msDFMuX, S=11, R=21, A=TCP $h
```

```
> 3,0 amm@mailhost
```

Use the host alias **mailhost**, rather than explicitly using the host name of this mail server. Either has the same effect.

```
rewrite: ruleset 3 input: "amm" "@" "mailhost"
rewrite: ruleset 6 input: "amm" "<" "@" "mailhost" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 0 input: "amm" "<" "@" "LOCAL" ">"
rewrite: ruleset 30 input: "amm"
rewrite: ruleset 3 input: "amm"
rewrite: ruleset 3 returns: "amm"
rewrite: ruleset 0 input: "amm"
rewrite: ruleset 9 input: "amm"
rewrite: ruleset 9 returns: "amm"
rewrite: ruleset 0 returns: $# "local" $: "amm"
rewrite: ruleset 30 returns: $# "local" $: "amm"
rewrite: ruleset 0 returns: $# "local" $: "amm"
```

sendmail recognizes that **mailhost** is a name for the current computer. It removes the **@mailhost** specification and runs the address back through Rulesets 3 and 0, resolving the address to the local mailer.

> **3,0 amm@cockatoo**

Recipient address explicitly to another host on the network.

```
rewrite: ruleset 3 input: "amm" "@" "cockatoo"
rewrite: ruleset 6 input: "amm" "<" "@" "cockatoo" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 3 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 0 input: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 0 returns: $# "ether" $@ "cockatoo" $: "amm" "<" "@" "cockatoo" ">"
```

The mail is sent to **cockatoo** using the **ether** mailer.

> **3,1,11,4 tracy**

The **ether** mailer defines the sender ruleset as 11.

```
rewrite: ruleset 3 input: "tracy"
rewrite: ruleset 3 returns: "tracy"
rewrite: ruleset 1 input: "tracy"
rewrite: ruleset 1 returns: "tracy"
```

```
rewrite: ruleset 11 input: "tracy"
rewrite: ruleset 11 returns: "tracy" "<" "@" "mailbox" ">"
```
Ruleset 11 adds the host name of this computer.
```
rewrite: ruleset 4 input: "tracy" "<" "@" "mailbox" ">"
rewrite: ruleset 9 input: "tracy" "<" "@" "mailbox" ">"
rewrite: ruleset 9 returns: "tracy" "<" "@" "mailbox" ">"
rewrite: ruleset 4 returns: "tracy" "@" "mailbox"
```
Since you used an explicit host name in the message address, the return address (the sender's address) should have the explicit host name of the sending computer.

```
> 3,2,21,4 amm@cockatoo
```
Ruleset 21 is specified as the reipient address ruleset in the **ether** mailer definition.
```
rewrite: ruleset 3 input: "amm" "@" "cockatoo"
rewrite: ruleset 6 input: "amm" "<" "@" "cockatoo" ">"
rewrite: ruleset 6 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
```
Known computer on the local network recognized, and .**LOCAL** appended.
```
rewrite: ruleset 3 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 2 input: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 2 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 21 input: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 21 returns: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 4 input: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 9 input: "amm" "<" "@" "cockatoo" "." "LOCAL" ">"
rewrite: ruleset 9 returns: "amm" "<" "@" "cockatoo" "." "rhino" "." "com" ">"
```
Ruleset 9 converts the **LOCAL** pseudodomain to the actual domain.
```
rewrite: ruleset 4 returns: "amm" "@" "cockatoo" "." "rhino" "." "com"
```

Configuration Files on NeXT Computers

Three different **sendmail** configuration files are shipped with NeXT computers. Each is used in a different

situation:

- **sendmail.subsidiary.cf**ÐFor use on a computer that has local mail spool storage but doesn't have complete mail routing information; it isn't a mail client or mail server. This is the default configuration file.
- **sendmail.mailhost.cf**ÐFor use on a mail server.
- **sendmail.sharedsubsidiary.cf**ÐFor use on client computers that import the **/usr/spool/mail** directory from a server.

Macros

The following table describes how macros are defined and used in the standard **sendmail** configuration files on NeXT computers.

Macro	Use	Definition	Where Used
M	Relay mailer	uucp ether etherl	mailhost subsidiary sharedsubsidiary
R	Relay	hostmail-relay mailhost mailhost	mailhost subsidiary sharedsubsidiary
V	Configuration file version	NX3.0M NX3.0S NX3.0X	mailhost subsidiary sharedsubsidiary
Z	Canonical address map	mail.byaddr	mailhost

e	SMTP greeting	$j Sendmail $v/$V ready at $b	All
j	Host name	$?m $w.$m $| $w $.	All
l	From header format	From $g $d	All
n	Mailer agent name	Mailer-Agent	All
o	Operators in addresses	.:%@!^=/[]	All
q	Format of sender's address	Dq$?x$x $.<$g>	All

Some of these macros are defined the same way in all the configuration files, some are defined differently in each file, and others are only defined in some files.

Note: The **$?** metasymbol denotes an **if** statement; the **$.** metasymbol denotes the end of an **if** statement; the **$|** metasymbol denotes an else statement. Thus **Dq$?x$x $.<$g>** can be read ªdefine the sender's address (**Dq**), if the full name is present (**$?x**), put in the full name with a space character (**$x**), end of the **if** construct (**$.**), put in a begin angle bracket (**<**), put in the sender's address (**$g**), put in the end angle bracket (>)º. Similarly, the construct **$?x$x$|$g$.** reads ªif there's a full name (**$?x**), put it in (**$x**), else (**$|**) put in the sender's address (**$g**), end of the **if-else** construct (**$.**)º.

The **Z** macro lists the name of an NIS map to be used for ªreverse aliasº translation. For example, if a user with the login name **rkabir** sends mail, the map referenced by the **Z** macro will contain an entry translating **rkabir** to another name, such as **Randy_Kabir**. If NIS is not running, or the referenced map isn't defined, no translation will occur (and no error will result).

The **o** macro defines the operators used in addresses. Operators separate tokens from each other.

Tip: If you change your configuration file, also make a small change to the definition of the **V** macro to indicate that the file has been modified. This is a good way for you to track changes, and can help other people recognize

changes.

The next table describes sample values for those macros that are defined in terms of other macros.

Macro	Sample Value
e	rhino.Tute.EDU Sendmail NX5.67c/NX3.0X ready at Wed, 15 Jul 92 14:03:06 -0700
j	rhino.Tute.EDU
l	From amm Fri Jan 3 13:54:40 1992
q	amm

Classes

The following table describes the classes defined in the configuration files. Remember that a macro and a class may have the same name but have completely different values and they are used in completely different ways. However, it's much easier to understand a configuration file if macros and classes with the same name are used in similar ways.

Class	Use	Definition	Where Used
R	Relay host	mail-relay	mailhost
		mailhost	subsidiary
		mailhost	sharedsubsidiary
V	UUCP partners	Output of **/usr/bin/uuname**	All

Notice that the **R** class is defined with the same value as the **R** macro. Both store the name (or names, in the case of the class) of the mail relay host. The **R** class defines all the names of the relay host, including aliases, and the **R** macro defines the preferred name of the relay host. Both are needed, because each is used differentlyÐclasses are lists of equivalent things, macros are variables used for simple substitution.

You can use the **R** class to determine whether a host is ªequivalentº to the relay host by using the **$=** metasymbol in the left-hand side of a rule. If the host is equivalent, you might want to process the mail differently. (Note that the **R** class, though defined, is not used in the default configuration files.)

Address Resolution

Mail addresses can be purely local (for example, **user**), can explicitly name another local host (for example, **user@host**), or can be remote (for example, **user@host.domain**). The following table shows how the various addresses are resolved for each of the three configuration files.

Address	**mailhost**	**subsidiary**	**sharedsubsidiary**
user	local	local	etherl to mailhost
user@host	ether	ether	etherl to mailhost
user@host.domain	ddn	ether to mailhost	ether to mailhost

Examine the line for **user@host.domain**. If this address is resolved on the mail server (using **sendmail.mailhost.cf**), the **ddn** mailer will be used. If the address is resolved on a computer using **sendmail.subsidiary.cf**, it will be forwarded using the **ether** mailer to the computer called **mailhost**. It will also be sent using the **ether** mailer to **mailhost** if resolved on a mail client using **sendmail.sharedsubsidiary.cf**.

Rulesets

The configuration files on a NeXT computer use the various rulesets in a consistent mannerÐRuleset 5 in **sendmail.mailhost.cf** does the same thing as Ruleset 5 in **sendmail.sharedsubsidiary.cf**. The following three tables summarize the uses of the various rulesets. The rulesets are divided into three groups: the standard rulesets, the nonstandard rulesets that aren't mailer-specific rewriting rulesets, and the mailer-specific rewriting

rulesets. The following table describes the standard rulesets.

Ruleset	Purpose
0	Mailer resolution
1	Standard sender header address preprocessing
2	Standard recipient header address preprocessing
3	Canonicalization and focusing
4	Standard header address postprocessing

The following table shows rulesets that aren't standard rulesets and aren't used by any of the mailers for address rewriting. These rulesets are usually called by other rulesets to perform frequently used operations.

Ruleset	Purpose
5	Converts **user@host.uucp** to **host!user**
6	Implementation of various local conventions
9	Address cleanup prior to passing address to a mailer
30	Runs Ruleset 3 followed by Ruleset 0

The next table summarizes the mailer-specific rewriting rulesets. Ruleset 12 is used only by mail clients, and Ruleset 22 is used only by mail servers and mail clients (it isn't used in **sendmail.subsidiary.cf**).

Ruleset	Purpose
10	Sender address rewriting ruleset for **local** and **prog** mailers
20	Recipient address rewriting ruleset for **local** and **prog** mailers
11	Sender address rewriting ruleset for **ether** mailer
21	Recipient address rewriting ruleset for **ether** mailer
12	Sender address rewriting ruleset for **etherl** mailer
22	Recipient address rewriting ruleset for **etherl** mailer (mail client) Sender and recipient address rewriting ruleset for **ddn** mailer (mail server)

13 Sender address rewriting ruleset for **uucp** mailer

23 Recipient address rewriting ruleset for **uucp** mailer

Tricks of the Trade

Supporting **sendmail** is a challenging task not to be undertaken by the fainthearted. This section provides some general suggestions for **sendmail** success.

General Tricks

- Consider finding a mentor to help you learn about **sendmail**. Look for someone who has experience with **sendmail** and wouldn't mind answering questions.
- If you want to explore **sendmail** in depth, source code is potentially available by anonymous FTP from, for example, **uunet.uu.net** (in **/mail/sendmail**).

Warning: Any general **sendmail** source code available from the archives is unlikely to work as is on a NeXT computer. Various NeXT-specific modifications have been made, particularly to support NetInfo.

- You might find some of the discussions that occur on the UseNet newsgroup **comp.mail.sendmail** helpful. In addition, other readers might be able to answer your questions or provide assistance.

Operational Tricks

There are three important operational tricks that can come in handy in specific situations.

Mail to Aliases Including the Sender

When a user sends mail to a mail alias that, when resolved, includes the sender's account name, the sender normally doesn't receive a copy of the mail. However, if the **m** option is set in the **sendmail** configuration file, the sender *will* receive a copy of the message in this situation. It's important to note that the setting in the **sendmail** configuration file on a mail client isn't significantÐit's the setting in the configuration file on the mail server that determines whether the sender will receive a copy of the mail message.

Including realname in the From: Address

The definition of the **q** macro in the sender's configuration file determines the format of the **From:** address in the header. The definition in **sendmail.sharedsubsidiary.cf** does not include the **realname** field. If you want to include the **realname** field in the **From:** address, modify the definition to match the following:

```
Dq$?x$x <$g>$|$g$.
```

With this definition, the format of the return address will be similar to the following, assuming the **realname** field value is present):

```
Tracy Q. Roykirk <troykirk>
```

If the full name is not present, the format of the return address will be similar to the following:

```
Tracy Q. Roykirk <troykirk>
```

If you want to begin with the sender's address and then include the **realname** field in the **From:** address, modify the definition to match the following:

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

With this definition, the format of the return address will be similar to the following:

```
troykirk (Tracy Q. Roykirk)
```

Mail to Unknown Domains

As described earlier, the **M** macro, **R** class, and **R** macro are used to define the relay mailer and relay computer. In the configuration file **sendmail.mailhost.cf**, these are defined as **uucp**, **mail-relay**, and **mail-relay**. By default, these values aren't used. Instead, mail to an unknown domain is sent to the **ddn** mailer, directly to the unknown domain. If your mail server isn't connected to that domain, the mail will be returned as undeliverable.

If you want mail to an unknown domain to be handled differently, define the **M** macro, **R** class, and **R** macro with appropriate values in **sendmail.mailhost.cf**. Then, in the same file, modify Ruleset 0. The lines to be changed begin at line 338 in version NX3.0M:

```
# If you want to pass all other explicit domain names up the ladder
# to our forwarder then uncomment the following line.
#R$*<@$*.$+>$*   $#$M $@$R  $:$1<@$2.$3>$4  user@any.domain
# and comment out this one.
R$*<@$+.$->$*    $#ddn $@ $2.$3 $:$1<@$2.$3>$4    user@any.domain
```

Make the changes described in the comments. The result will look like this:

```
# If you want to pass all other explicit domain names up the ladder
# to our forwarder then uncomment the following line.
R$*<@$*.$+>$*    $#$M    $@$R $:$1<@$2.$3>$4    user@any.domain
# and comment out this one.
#R$*<@$+.$->$*    $#ddn $@ $2.$3 $:$1<@$2.$3>$4    user@any.domain
```

Debugging Tricks

When debugging a **sendmail** configuration problem, you might want to examine the values of various macros in order to determine exactly what's defined and how. You can make additions to your configuration files specifically for this purpose.

Adding a Ruleset to Examine Macros

The following definition for Ruleset 16 uses the **error** mailer to display the values of the **j** and **m** macros.

```
S16
R$*     $#error $:j=$j, m=$m
```

Run **sendmail** in address test mode and invoke Ruleset 16:

```
> 16 test
rewrite: ruleset 16   input: "test"
rewrite: ruleset 16 returns: $# "error" $: "j" "=" "cockatoo" "." "rhino" "." "com" "," "m"
"=" "rhino" "." "com"
```

Normally, the **error** mailer is only invoked in Ruleset 0. However, in this case it provides a simple message printing mechanism. The **j** macro has the value **cockatoo.rhino.com**, and the **m** macro has **rhino.com** as its value.

Adding Header Lines to Examine Macros

Some macros don't have meaningful values until a message has been processed, such as **u** (the recipient's address), or **q** (the format of a sender address). Using a ruleset to examine these macros won't work. Instead, you can modify the headers to display these values.

Add the following lines to the collection of header definitions:

```
HX-Test1: j=$j, m=$m
HX-Test2: u=$u, q=$q
```

Preceding the name of a nonstandard header line with **X-** follows mail header format conventions, and makes sure that your message complies with all relevant standards.

Send a small test message to yourself. You'll see two extra header lines, like this:

```
X-Test1: j=cockatoo.rhino.com, m=rhino.com
X-Test2: u=tr, q=tracy
```

The message was sent to the alias **tr** by the user **tracy**.

Viewing All the Headers

NeXT's Mail.app application shows only some of the header lines in a message. If you're using the Mail application from Release 3.3 or later, you can use the Show All Headers command from the Message menu to examine all the headers. If you're using a previous version of Mail and the headers you want to see are not visible in the message window, you can view the full headers of the message by performing the following.

1. Click the message heading in the mailbox window.
2. Copy the message with Command-c *without* clicking in the message area.
3. Paste the message into, for example, a new Edit window.

If the message was just ASCII text, you'll see the complete message in the Edit window. You'll see only the headers if the message contained rich text, an attached document, voice, or an image.

Here's an example of the complete headers of a message that was sent by Tracy Roykirk from the computer **cockatoo**. It was forwarded to **mailbox**, and there delivered to the user **amm**.

From Tracy_Roykirk Wed Jul 15 13:57:41 1992
Next-Reference: Testing.attach, 1/1

Return-Path: <Tracy_Roykirk>
Received: from cockatoo.Rhino.COM (cockatoo) by mailbox.Rhino.COM (NX5.67c/NX3.0M)
id AA00320; Wed, 15 Jul 92 13:57:35 -0700
From: Tracy_Roykirk (Tracy Q. Roykirk - SysAdmin)
Message-Id: <9112270031.AA00320@mailbox.Rhino.COM>
Received: by cockatoo.rhino.com (NX5.67c/NX3.0X)
id AA00937; Wed, 15 Jul 92 13:57:30 -0700
Date: Wed, 15 Jul 92 13:57:35 -0700
Received: by NeXT.Mailer (1.85)
Received: by NeXT Mailer (1.85)
To: amm
Subject: Testing