Sun™ Internet Mail Server™ 3.5 Reference Manual

THE NETWORK IS THE COMPUTER™

A Sun Microsystems, Inc. Business
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300   fax 650 969-9131

Part No.: 805-4379-10
Revision 10, September 1998

Contents

Figures

Tables

Preface

Sun™ Internet Mail Server™ 3.5 (SIMS 3.5) is an enterprise-wide, open standards-based, scalable electronic message handling system. The *Sun Internet Mail Server 3.5 Reference Manual* provides reference information about the Sun Internet Mail product.

This guide should be used as a companion to the *Sun Internet Mail Administrator's Guide*. The administrator's guide focuses on how to configure, maintain, monitor, and troubleshoot Sun Internet Mail using the Administration Console. The *Sun Internet Mail Server 3.5 Reference Manual* provides information on command line utilities and configuration files. This information enables you to configure, maintain, monitor, and troubleshoot Sun Internet Mail.

Who Should Use This Book

This book is intended for two audiences:

- Highly technical network administrators who are experienced in working with Solaris™ systems and who manage a network comprised of Sun™ workstations, personal computers (PCs), Macintoshes, or IBM mainframes that share resources. This network administrator has previous experience planning, installing, configuring, maintaining, and troubleshooting an enterprise email system.
- Moderately technical network administrators with some Solaris experience who manage a network that includes Sun workstations, PCs, and Macintoshes that share resources. This network administrator may not have previous experience planning, installing, configuring, maintaining, and troubleshooting an email system.

How This Book Is Organized

Chapter 1, "Overview," describes the overall Sun Internet Mail product and its components. It describes the features of Sun Internet Mail and provides a functional overview of the product.

Chapter 2, "Commands Reference," is a complete reference to the server-side utilities used to configure and administer the Sun Internet Mail product. The commands are listed by component. This chapter describes what each command and its options can do and provides examples.

Chapter 3, "IMTA Configuration," describes IMTA configuration files that you can edit and that are supported by Sun Internet Mail.

Chapter 4, "Sun Directory Services Configuration," describes the Sun Directory Services configuration files.

Chapter 5, "Message Access and Store Configuration" describes the Message Access and Message Store configuration file.

Chapter 6, "System Architecture," describes each component of the Sun Internet Mail.

Appendix A, "Supported Standards," lists all the industry standards that are supported by Sun Internet Mail.

Related Information

The following books are related to Sun Internet Mail. Included in this documentation set are:

- *Sun Internet Mail Server 3.5 Advanced Installation Guide*–Describes the planning and installation procedures for the Sun Internet Mail Server (SIMS) 3.5 software on Solaris SPARC and Intel-based x86 systems. In particular, it describes the installation of the software using the Graphical User Interface (GUI).
- *Sun Internet Mail Server 3.5 Administrator's Guide*–Describes how to fine-tune the default configuration, maintain, monitor, and troubleshoot your mail server using the Administration Console, a GUI.
- *Sun Internet Mail Server 3.5 Reference Manual* (this guide)–Provides detailed information on command line options, administrator-editable configuration files, system architecture, supported standards, and location of software files.

- *Sun Messaging Connectivity Services cc:Mail Channel Guide*, *Sun Messaging Connectivity Services Microsoft Mail Channel Guide*, and *Sun Messaging Connectivity Services PROFS Channel Guide*—These guides describe how to fine-tune your configuration, in order to maintain, monitor, troubleshoot, and integrate your LAN-based proprietary mail systems with SIMS 3.2.
- *Reference manual pages* (*man pages*)—Describe command-line utilities and detailed information about the arguments and attributes relevant to each command.
- Sun Internet Mail Web site (located at `http://www.sun.com/sims`) offers up-to-date information on a variety of topics, including:
  - Online product documentation and late-breaking updates
  - Data sheets and evaluation guide
  - Technical white papers
  - Product demos
  - Press coverage and customer success stories
  - Client solutions

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Changes in Text

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output is printed using `courier` font.	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`

`AaBbCc123`	What you type, contrasted with on-screen computer output is printed using **`bold courier`** font.	`machine_name%` **`su`**
		`Password:`

AaBbCc123	Command-line placeholder; replace with a real name or value are printed using *italic* text.	To delete a file, type: `rm` *filename.*

TABLE P-1 Typographic Changes in Text

Typeface or Symbol	Meaning	Example
AaBbCc123	Book titles, new words or terms, or words to be emphasized are printed using *italic* text.	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Shell Prompts in Command Examples

The following table shows the default system and superuser prompts for the C, Bourne, and Korn shells.

TABLE P-2 Shell Prompts in Command Examples

Shell	Prompt
C shell user prompt	`machine_name%`
C shell superuser (`root`) prompt	`machine_name#`
Bourne shell and Korn shell user prompt	`$`
Bourne shell and Korn shell superuser (`root`) prompt	`#`

Note – Although the majority of commands can be run without special superuser permissions, some commands can only be performed as `root`; these commands include: `imta dirsync, imta start, imta stop,` and `imta restart.` Other commands that require `root` privileges are noted within the document.

CHAPTER **1**

Overview

This chapter describes the overall Sun Internet Mail Server 3.5 (SIMS) product, main components, and key features.

What Is Sun Internet Mail Server?

Sun Internet Mail Server is an extensible framework of independent modules that create an enterprise-wide, open standards-based, scalable electronic message handling system.

A message handling system is the combination of message user and transfer agents, message stores, and access units that together provide electronic mail.

The components of this message handling system are:

- **Message Access and Store**. Message Access and Message Store are the repositories of user messages, and the means to retrieve and process those messages. Sun Internet Mail Server supports both the Internet Message Access Protocol version 4 (IMAP4) and the less flexible but widely-implemented Post Office Protocol version 3 (POP3).

  The primary message store for Sun Internet Mail Server is the Sun Message Store. The mail server also retains support for the Solaris Mailbox Format Store to ease migration for sites with an installed base of traditional `/var/mail` clients.
- **Internet Message Transfer**. The Internet Message Transfer Agent (IMTA) is responsible for the routing, transfer, and delivery of Internet mail messages. The Sun Internet Mail Server includes a fast, scalable, and flexible IMTA. This replaces the Sendmail utility that is bundled with most UNIX® systems and was used in Sun Internet Mail Server 2.0.

- **Sun Directory Services**. The directory is the central repository for meta-information: user profiles, distribution lists, and other system resources. Sun Internet Mail Server is bundled with a dedicated Lightweight Directory Access Protocol (LDAP) directory service, but is specifically designed to use any directory service compatible with LDAP version 2.
- **Administration Services**. The administration of Sun Internet Mail Server is a comprehensive GUI-based installation, configuration, and monitoring environment. Sun Internet Mail Server is based on the Java™ Management Application Program Interface (JMAPI) framework.
- **Sun Messaging Connectivity Services**. Provides batch-mode connectivity to proprietary message transfer systems. These include the "LAN Mail" systems Lotus cc:Mail and Microsoft Mail, and the mainframe-base IBM OfficeVision (PROFS).

Refer to Chapter 6, "System Architecture" for detailed descriptions of each component.

Key Features

Key features of the product are:

- Client/server architecture
- High performance and high scalability
- Internet open standards
- Native support for nomadic and disconnected operation
- Integrated directory service
- Dependable transfer and delivery
- Support for non-Internet email systems
- Comprehensive easy-to-use GUI-based administration

The consolidated Sun Internet Mail Server components are comprised of two main product packages:

- Departmental Edition
- Enterprise Edition

Departmental Edition

The *Departmental Edition*, also referred to as the Sun Internet Mail Server - Departmental Edition, is intended for local departmental environments that do not need the scalability and extensive configurability of an enterprise server. The departmental package performs its own routing and delivery within a local office or department, but hands off interdepartmental mail to an enterprise or backbone server. The package is simple to install and configure, and requires minimal operator intervention. The key features of the Sun Internet Mail Server - Departmental Edition are:

- Multithreaded IMAP4 and POP3 servers. These are optimized for up to 500 simultaneously connected IMAP users.
- Internet Message Transfer Agent (IMTA). The IMTA is restricted to two external Simple Mail Transfer Protocol (SMTP) channels. One channel is dedicated to the local intranet, and the other to a "smart host" backbone or firewall connection.
- Server daemon processes that are spawned by the multiprocessing `inetd` utility instead of the Dispatcher.
- Multiprocessing LDAP directory service.
- Full featured Sun Message Store (SMS).
- Centralized GUI Administration Console.
- Interface to Solaris Mailbox Format (`/var/mail`) message store.
- Supports Internet standard mail protocols.
- Integrated backup and restore.
- SSL protocol security for IMAP and POP servers.
- IMAP and POP proxy daemons.
- MAPI providers.
- HotJava Views and Sun Web Access clients.

Enterprise Edition

The Enterprise Edition, also referred to as the Sun Internet Mail Server - Enterprise Edition, provides a full featured messaging server for large user communities, enterprise backbone management, and Internet firewall applications. The key features of the Sun Internet Mail Server - Enterprise Edition include all the features of the Departmental Edition plus the following:

- Multithreaded IMAP4 and POP3 servers. These servers have higher performance and a much smaller footprint than the multiprocessing servers in the standard package. These servers support as many as 10,000 simultaneous connections and 100,000 mailboxes on an E3000-class server.

- Full-featured IMTA. This includes extensive address rewriting and channel management facilities. Multithreaded Job Controller scales to the number of licensed users.
- Multiprocessing LDAP directory service.
- Server daemon processes that are managed by the Dispatcher instead of `inetd`, for better scalability.
- Pipe channels that support extensibility of the IMTA through native UNIX system scripts.
- High performance databases that replace internal flat files for the user cache, distribution lists, mappings, and forwarding.
- Multiprocessing hardware support.
- Anti-spamming.
- Anti-relay.
- Multiple configurable channels.
- Sun SDK APIs available for custom channel development.
- SMCS (Sun Messaging Connectivity Services), for the Microsoft Mail, cc:Mail, and PROFS channels.
- Asymmetric HA.
- MAPI providers.
- HotJava Views and Sun Web Access clients.

CHAPTER 2

Commands Reference

This chapter contains the command-line administration utilities used to configure and administer Sun Internet Mail. The following sections describe how the commands are organized and how to use them. This chapter contains commands for:

- Message Access and Store
- Sun Directory Services
- Internet Message Transfer Agent (IMTA)
- Security and Authentication
- Installation
- Sun Messaging Connectivity Services

Detailed descriptions of these commands are contained in the reference manual pages (man pages).

How the Commands Are Organized

The commands are organized as follows:

- Commands are grouped by system component, which includes:
  - Message Access and Store utilities
  - Directory Services utilities
  - Internet Message Transfer Agent (IMTA) utilities
  - Security and Authentication
  - Installation
  - Sun Messaging Connectivity Services
- Commands are listed alphabetically within each component section.
- Headings identify the commands.

- Brief overview of the command includes any attributes that can be modified.
- Subsections describe the tasks performed by the command options, display the command format, and provide examples of the command usage.

Each example shows how the command might be used and how the executed actions can be verified. Because some commands do not display any output, another command is used to read or list the modified attribute values. Detailed man pages are available for additional information about each command.

Using the Commands

These command-line utilities allow you to configure and manage server resources from the Sun Internet Mail. You must be logged in as `root` to execute administrative utilities. The command descriptions will specify any special privileges. In the examples shown in this chapter, the commands requiring `root` privileges are shown with a # prompt and those *not* requiring `root` privileges are shown with a `%` prompt.

Most of the utilities are located in `/opt/SUNWmail/sbin`. You can add the location of the utilities in your `PATH=` environment variable. This step simplifies using the utilities so that you can enter just the name of the command (as shown in all the examples) instead of its full path.

Message Access and Store

Message Access and Store refers to the data stores, protocol servers, software drivers, and libraries that support message delivery, storage, retrieval, and final disposition. The following command line utilities are used for message access and storage and are outlined in this section. Detailed information about access and store utilities can be found in the man pages.

`imaccessd`

`imaccessd` provides email clients with access to the Sun Internet Mail Server. The `imaccessd` daemon supports two access protocols: Post Office Protocol, version 3 (POP3, RFC 1939); and Internet Message Access Protocol (IMAP, RFC 2060). The `imaccessd` daemon process normally runs whenever the mail server is up. Unlike other commands, `imaccessd` is a daemon which, when started, runs in the background. If this daemon is not running, all client requests for IMAP or POP connections receive a “Connection Refused” error.

Note – You must be logged in as `root` to use `imaccessd`.

Syntax

```
imaccessd [ -d ][-l config_file_location][ -p service=port ][ -t thread_limit ][ -v ]
```

The following options are used with `imaccessd`:

`-d`	Turn debugging mode on.
`-l` *config_file_location*	Location of the Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.
`-p` *service=port*	Specifies the port to be monitored. *service* must be either `pop3`, `imap`, `pop3s`, or `imaps` (SSL); *port* must be a numerical TCP port number. If the `-p` option is not specified, the monitoring defaults to support both IMAP and POP3 on their standard services port numbers. Multiple `-p` options can be used. The ports specified by the `-p` option override the defaults in `/etc/services`. When *port*=0 the service should be disabled.
`-t` *thread_limit*	Specifies the maximum number of threads per process. *thread_limit* defaults to 50, and rarely needs to be redefined.
`-v`	Write detailed (verbose) diagnostic messages to the debug log.

In the event that a fault occurs in a thread in `imaccessd`, this thread is terminated and a timestamped core file is created in `/var/opt/SUNWmail/ims/adm/corefiles`. This core file should be sent to SunSoft to help diagnose the problem that occurred and to improve the reliability of future versions of SIMS. Up to 10 core files can be created during the lifetime of the `imaccessd` parent process.

After a fault in a thread, the files opened by this thread are closed, and the client whose connection was terminated should be able to reconnect immediately. To limit side effects of the fault, the process containing the faulted thread continues with the other threads, but is not used for new incoming connections.

The `imaccessd` process should never be killed using the `kill -9` command. To kill the process, use the `kill` command without the `-9` argument—this kills the parent process. If `kill -9` is used, run `imcheck -c` before restarting `imaccessd`.

Note – `imaccessd` is normally started and stopped with the `im.server` script (`/etc/init.d`)

Examples

To provide normal IMAP service only on Sun's usual IMAP debugging port, use the following command:

```
# imaccessd -p imap=145
```

The following command will disable IMAP:

```
# imaccessd -p imap=0
```

imbackup

`imbackup` is the utility used to back up stored messages.

Syntax

```
imbackup -f ( device | file | - ) | -n[ -d datefrom ]\
[ -b blocking_factor] [ -a ] [-l config_file_location] [ -u usernames_file | userlist ]
```

The following options are used with `imbackup`:

`-f`	Specify the file name or device to which the backup is written. For example:
	`-f ./backup.0515`
	`-f /dev/rmt/0`

`-f -`	This option writes to `stdout` (to pipe the backup to another tool.
`-n`	This option is for simulation only; nothing is written.
`-a`	Autoload tapes when end-of-tape is reached, for use with autoloading tape drives.
`-d` *datefrom*	This option specifies the date from which messages are to be backed up, expressed as `yyyymmdd` (see the following example). The default is to back up all messages, regardless of their date.
`-b` *blocking_factor*	This option indicates the blocking factor; all actual writes on the backup device are performed by blocks of the size 512 bytes x `blocking_factor`. The default is 20.
userlist	Used to back up only specified users. By default, all users are backed up.
`-u` *usernames_file*	Used exclusively with *userlist* to back up only some users whose names are in the file *usernames_file*.
`-l` *config_file_location*	Location of the Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Examples

The following examples show two different ways to use the `-f` option:

```
# imbackup -f ./backup.0515
# imbackup -f /dev/rmt/0
```

The following example describes usage of the `-d` option to specify a particular date from which to backup messages. In this example, a message store backup from 5/1/97 (specified *yyyymmdd*) to the present is performed.

```
# imbackup -d 19970501 -f /dev/rmt/0
```

imcheck

The `imcheck` command validates the message store and the user files, reports errors, and generates message store reports. In addition to validating the message store and generating reports, it also allows you to recover the message store from a crash.

Messages may be lost if a crash occurs after the messages have been removed from mail queue by IMTA, but have not yet been "sync-ed" in the user file. When the `-c` option is specified, `imcheck` looks at all the messages delivered to the message store within the last few minutes before the crash, verifies if they are in the user files, and redelivers those that are not. Users may get the same message twice after a crash recovery.

Note – You must be logged in as the message store owner to use this utility.

Syntax

```
imcheck -d [date] [-f filename] [-l config_file_location]
imcheck -r [date] [-f filename] [-l config_file_location]
imcheck -u [user] [-f filename] [-l config_file_location]
imcheck -c [-f filename] [-l config_file_location]
imcheck -t [-l config_file_location]
```

Except for the `-f` and `-l` options, each option must be specified in a separate command. In other words, you cannot combine the `-d`, `-r`, `-u`, `-c`, and `-t` options together in one command.

The following options are used with `imcheck`:

`-d` [*date]*	Generates a detailed store report. If a specific date is given (in the form yyyymmdd), `imcheck` generates a report for the specified date.
`-r` [*date]*	Generates a regular report. If a specific date is given (in the form yyyymmdd), `imcheck` generates a report for the specified date.
`-c`	Perform a crash recovery. This selection can only be used after a crash. It must be run before the IMTA has started and should *not* be running with IMTA at the same time.
`-f` *filename*	Redirect all output to the file *filename*. The file is placed in the current working directory. It is *not* placed under `ADM_ROOT`. *filename* can be an absolute path.
`-u [`*user*`]`	Generate a user report for all message store users. If a specific user name is given, `imcheck` generates a user report for the specified user; otherwise, a report for all users is generated.
`-t`	Check if the message store has potentially been corrupted.
`-l` *config_file_location*	Location of the Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Note – When using the `-u` and `-c` options, imcheck can only be run when the message access is stopped.

Example

The following example generates report for the user “Jane” and directs the output to a file named “outputfile.”

```
# imcheck -u Jane -f outputfile
```

The following output is an example of a regular report (`imcheck -r`):

```
verbose store report with verification on Fri Oct 10
15:54:58 1997

Message data report
 Data for 1 year(s) follows:
1997  30 day(s), 3 bucket(s) per day

        1001 n-recs  end-of-file  verify_data
        001     394     17958940           ok
        002     397     13822883           ok
        003     399      7578785           ok

        1002 n-recs  end-of-file  verify_data
        001     833     23289349           ok
        002     864     12327173           ok
        003     812     21576875           ok

        1003 n-recs  end-of-file  verify_data
        001     873     16900476           ok
        002     878     21127935           ok
        003     781     23281795           ok

.
.
.
Index files report

Data for 1 years(s) follows:
    1997  30 day(s),
n-bkts complexity n-entries next-entry verify_links
ParseLevel  Version

   1001    3 COMPLETEPARSE    1190     1191       ok
3    0.0
   1002    3 COMPLETEPARSE    2509     2510       ok
3    0.0
   1003    3 COMPLETEPARSE    2532     2533       ok
3    0.0
   1004    3 COMPLETEPARSE    1472     1473       ok
3    0.0
   1005    3 COMPLETEPARSE       0        1       ok
3    0.0
.
.
.
```

The following output is an example of a detailed report (`imcheck -d`):

```
verbose store report with verification on Fri Oct 10 15:55:18 1997

Message data report
  Data for 1 year(s) follows:
     1997  30 day(s), 3 bucket(s) per day
1001 n-recs  end-of-file  verify_data
        001     394     17958940           ok
        002     397     13822883           ok
        003     399      7578785           ok

        1002 n-recs  end-of-file  verify_data
        001     833     23289349           ok
        002     864     12327173           ok
        003     812     21576875           ok

        1003 n-recs  end-of-file  verify_data
        001     873     16900476           ok
        002     878     21127935           ok
        003     781     23281795           ok
Index files report

Data for 1 years(s) follows:
    1997  30 day(s),

********1001*************************************

MAX_ENV_SIZE = 32768
MAX_BODY_SIZE = 262144
IDIR RECORD NO. 1 :
INDEXDIR: Index_Rec_Offset = 12
              INDEX_Rec_Size = 344
                     Ref_Cnt = 1
                      In_use = 1
                        CUID = 1.0
     INDEX:   Index_Rec_Size = 344
              Index_Rec_cksm = 0
                  Mdata_file = 1997/1001/003
                Mdata_offset = 12
                 Env_present = 1
                    Env_size = 198
                Body_present = 1
               Body_Rec_size = 60
MSGDATA:      Record_size = 4997
```

```
RFC821_rcpts = u4
Index_file = 1997/1001
                  Idir_entry = 1
                Message_size = 4930
                RFC821_rcpts = u3
IDIR RECORD NO. 2 :
  INDEXDIR: Index_Rec_Offset = 356
              INDEX_Rec_Size = 344
                     Ref_Cnt = 1
                      In_use = 1
                        CUID = 2.0
     INDEX:   Index_Rec_Size = 344
Index_Rec_cksm = 0
                  Mdata_file = 1997/1001/001
                Mdata_offset = 12
                 Env_present = 1
                    Env_size = 198
                Body_present = 1
               Body_Rec_size = 60
   MSGDATA:      Record_size = 2362
                  Index_file = 1997/1001
                  Idir_entry = 2
                Message_size = 2295

.
.
.
```

imdeluser

`imdeluser` is a utility for the system administrator to remove a user or public shared folder from the message store. `imdeluser` is a utility command and needs to be run on the server as `root`.

Syntax

```
imdeluser [-u file | - ] [ username... ]
```

The following options are used with `imdeluser`:

`[-u` *file* `| - ]`	Read user names from *file*. Use '-' to read the user names from `stdin`.
username	*username* is either a regular `userid` or a public shared folder name.

Examples

The following example deletes a user named "joe."

```
# imdeluser joe
```

The next example deletes a public shared folder.

```
# imdeluser #shared.ims-team
```

`imdeluser` prompts for the administrator's Distinguished Name (DN) and password:

```
# imdeluser joe
  DN:
  Password:
```

If all of the following conditions are valid, all the folders and user files for the specified user are removed from the message store:

- User entered the correct DN and password
- User or public shared folder exists in the message store
- User is not receiving messages

imexpire

`imexpire` scans all user folders in the message store and marks all the messages that match the specified criteria as permanently deleted, or "expired."

The deleted messages will be expunged from the user mailbox when the user connects or disconnects from the server.

The actual data will be removed from the message store when `impurge -a` is run after the `imexpire` utility.

`imexpire` must be run on the message store server by root or by the message store owner.

Note – `imexpire` does not remove expired messages from the message store. It only marks those messages as "expired." You must run `impurge -a` after you run `imexpire` to reclaim the disk space. When `imexpire` is used with the `-s` option, it marks the "unseen" messages as "pending" instead of "expired." Once a message is marked as "pending", imexpire will not expire the message. You must run `impurge -a` to clear the "pending" flag.

Syntax

```
imexpire [-d date | -n num] [-i] [-l config_file_location] [-s] [-v]
```

The options for this command are:

`-d` *date*	"Expires" all messages older than the date specified. *date* must be of the form *yyyymmdd*, where *yyyy* specifies the year, *mm* specifies the month, and *dd* specifies the day.
-n *num*	"Expires" all messages older than *num* days.
`-i`	Interactive mode. With this option, `imexpire` prompts once for confirmation before removing any messages.
`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.
`-s`	"Expires" only messages that have been "seen". The `-s` option must be run together with the `-d` option or the `-n` option.
`-v`	Verbose mode. Provides details of the `imexpire` operation.

Examples

The following command “expires” all messages from the message store that are older than February 14, 1998:

```
% imexpire -d 19980214
% impurge -a
```

imexportmbox

`imexportmbox` is a SIMS utility which allows the message store owner (usually `inetmail`) to copy a user’s folders to a target directory. Unless the `-s` option is used to specify a single folder, all the user’s folders are copied into the destination directory preserving any folder hierarchies in the form of directories. If the destination directory does not exist, `imexportmbox` will attempt to create it. If a file already exists in the destination directory, `imexportmbox` will not overwrite the file and will move on to the next folder.

`imexportmbox` must be run as the message store owner as specified in the `ims.cnf` file. The default owner should be set to `inetmail`. The destination directory must allow the message store owner write permission.

Syntax

```
imexportmbox -u user -d dest_dir [-s src_folder]
```

The options for this command are:

`-u` *user*	The user ID as used in the message store. This is a required option.
`-d` *dest_dir*	The destination directory where the folders will be created and written. This is a required option.
`-s` *src_folder*	The name of a single folder to export. This option is optional.

Examples

In the following example, the command will extract all email for user `smith1`. `smith1` is a valid user account in the SIMS message store. User `smith1` has three folders on the store: `INBOX` (the normal default user folder), `Mail/flying`, and `Mail/Christmas`. The destination directory will be `/tmp/joes_mail`, and it must be writable by the message store owner (`inetmail`). The command must be run as the message store owner.

```
% imexportmbox -u smith1 -d /tmp/joes_mail/
```

`imexportmbox` will then transfer each message store folder into a `/var/mail` conforming file. Thus you will get the following files:

- `/tmp/joes_mail/INBOX`
- `/tmp/joes_mail/flying`
- `/tmp/joes_mail/Christmas`

imimportmbox

As the administrator, to populate your SIMS 3.5 message store with a user's existing messages and folders, you need to execute the message store utility called `imimportmbox`. This utility helps you to move the user's existing inbox messages and folders from existing `/var/mail` format to the newly installed message store.

It is possible to specify a non-existent user with `imimportmbox`.

Note – Run this utility as `root` or as the message store owner.

Syntax

```
imimportmbox [ -u user ] [ -s src_folder ] [ -d dst_folder ] [ -l config_file_location]
```

The options for this command are:

`-u` *user*	User ID
`-s` *src_folder*	Source folder. *src_folder* can be inside `/var/mail`, but it does not need to be.
`-d` *dst_folder*	Destination folder
`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Examples

`imimportmbox` migrates the specified `/var/mail/`*folder* for the specified user to the new message store. If the destination folder is not specified, `imimportmbox` calls the destination folder by the same name as the source folder. In the following example, the command migrates the default `/var/mail INBOX` for the user `smith`, to the `INBOX`.

```
# imimportmbox -u smith -s /var/mail/smith -d INBOX
```

Similarly, if you are trying to move a folder called `test` from `/home/smith/folders/` to the SIMS 3.5 message store, use the following command:

```
# imimportmbox -u smith -s /home/smith/folders/test -d test
```

If a destination folder called `test` already exists in the SIMS 3.5 message store, `imimportmbox` asks if you want to append to the existing folder in the mailbox or abort the move process.

iminitquota

The `iminitquota` utility reinitializes the quota limit from the specified user's LDAP directory entry and recalculates the total amount of disk space that is being used by the specified user. It updates the file `quota` under the user's `Adm` directory in the Message Store. This file will be read by the delivery agent when trying to determine if a certain user is over-quota.

iminitquota must be run as the message store owner as specified in the ims.cnf file. The default owner should be set to inetmail.

Note – When the -a option is specified, iminitquota uses the MTA.hostname value specified in the /etc/opt/SUNWmail/admin/adminserver.properties configuration file as the mailHost attribute to search for local message store users in the LDAP directory.

Syntax

```
iminitquota [-l config_file_location] -a | -u username
```

The options for this command are:

-a	Initializes and updates the user quota files for every message store user on the server. You must specify either the -a or -u option.
-l *config_file_location*	Location of Message Access Services configuration file (ims.cnf). If the -l option is not specified, the default location (/etc/opt/SUNWmail/ims/) will be used.
-u *username*	Reinitialize the quota -elated information for this user. You must specify either the -a or -u option.

Example

The following example requests quotas information for a user named “Jane.”

```
# iminitquota -u Jane@XYZ.com
```

One of the following exit status codes from `sysexits.h` is returned:

`EX_OK`	Normal exit. Quota information successfully updated for the specified user.
`EX_USAGE`	The command line usage was incorrect.
`EX_NOUSER`	The specified user does not exist in the message store.
`EX_SOFTWARE`	`iminitquota` failed because of unexpected software error. This usually means fail to access the users's LDAP entry.
`EX_OSERR`	An operating system error occurred during the `iminitquota` process, most commonly out of virtual memory.
`EX_CONFIG`	Mail server configuration error. This usually means that the store has not been initialized.

impurge

The `impurge` utility removes messages from the Message Store that are no longer referenced from any user folders, and returns the space to the file system. When a user deletes a message, the reference to the message is also removed. Eventually, all users who received the message may remove their references. When the last reference is gone, the message can be purged from the store.

The purge operation requires a considerable amount of time and system resources. Do not wait until your disk is full before attempting a message purge. Run `impurge` while there is more disk space than the amount of space used by the message store on the busiest 24 hour period of the message store. You can check the message store disk usage by noting the disk usage increase on the `/var/opt/SUNWmail/ims` partition over a 24 hour period.

Note – Messages under 2 days old will not get purged.

Syntax

`impurge` [`-av`] [`-p` *percentage*] [`-s` *size*] [`-l` *config_file_location*]

The options for this command are:

`-a`	Do an exhaustive search to locate all messages that can be purged and purge all of them. Note that if the `-a` option is used, the `-p` and `-s` options *cannot* be used.
`-p` *percentage*	Percentage purge–If the total amount of disk space recovered by removing messages marked for deletion exceeds a specified percentage of the total message traffic received on a particular day, then a purge operation is performed. For example, if you specified 10 percent for the percentage purge option, each day a computation is performed to determine if the total amount of space recovered by removing messages marked for deletion exceeds 10 percent of total message traffic received on this particular day. If on a particular day, the total amount of space recovered equals 9.5 percent, then the purge operation is not performed. If a few days later, the total amount of space recovered equals 10.3 percentage, then the purge operation is performed.The intent of the percentage option is to speed up the purge process by not purging every day and each time that `impurge` is run, but only those days when enough space can be gained.
`-s` *size*	Perform the purge *only* for the days where the amount of space that would be reclaimed exceeds the specified *size*. The *size* can be entered in bytes or kilobytes, using the suffix `K`; megabytes, using the suffix `M`; or gigabytes, using the suffix `G`.
`-v`	Verbose–provides details of the purge operation; otherwise, `impurge` is silent unless an error occurs.
`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Examples

The following sample represents a small work group with 500 Mbyte of storage for user mailboxes that has decided to reclaim space every morning, but only if 10 Mbyte or more of disk space can be reclaimed.

```
% impurge -s 10M
```

For the next example, assume that a larger work group has 10 Gbyte of storage, and on a typical day 5 Mbyte of new mail is received, although this can vary. For efficiency, the site decides to use `impurge` every morning, but only if the reclaimed space is half as large as the particular day's mail messages. Using this formula, if that day's traffic is 40 Mbyte, then the `impurge` for that day must find at least 20 Mbyte to reclaim, or else `impurge` is skipped. This command line would look like the following:

```
% impurge -p 50
```

The percentage stipulation is used for performance reasons, and it would not be worthwhile to purge a day's data if the amount of that data was not significant.

Note – `impurge -a` will clear the `PENDING` flag set by `imexpire`.

---

imquotacheck

`imquotacheck`, the Quota Notification utility, calculates the total mailbox size for each user in the message store, compares the size with their assigned quota, and sends a notification via email to the users that have exceeded a set percentage of their assigned quota.

The default percentage used to determine quota is exactly 90%. The `-p` option may be used to specify a different percentage.

If the `-v` or `-u` options are not specified, `imquotacheck` displays only the users who have exceeded the quota.

`imquotacheck` must be run as the message store owner as specified in the `ims.cnf` file. The default owner should be set to inetmail.

Note – The content of the quota notification message cannot be changed.

Syntax

```
imquotacheck [-v] [-n] [-p percentage] [-u username] \
[-l config_file_location]
```

The following options are used with `imquotacheck`:

`-v`	Verbose mode. Prints the username, quota, total mailbox size, and percentage used of all users to stdout. If the `-u` option is specified, `imquotacheck` prints the statistics for the specified user only.
`-n`	Does not send notification to the users. Prints notification to stdout only.
`-p` *percentage*	Send a notification if the total mailbox size exceeds a specified percentage of the assigned quota.
`-u` *username*	Executes `imquotacheck` on a specified user. The user is displayed even if the quota has not been exceeded.
`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Examples

The following command sends a notification to all users that have a mailbox size greater than 80% of their assigned quota limit:

```
% imquotacheck -p 80
```

The following command checks the quota usage for each user in the message store without sending notification to the users:

```
% imquotacheck -v -n
```

imrestore

`imrestore` is the utility used to restore messages from the backup device into the message store.

Syntax

```
imrestore [ -b blocking_factor ] [ -t 1|2|3 ] [ -i ] [ -f device | file | - ] \
[ -c continuation_filename ] [ -l config_file_location] [ -u usernames_file | userlist ]
```

The following options are used with `imrestore`:

`-b` *blocking_factor*	This option indicates the blocking factor; all actual reads on the device are performed by blocks of the size 512 x blocking factor. The default is 20. Note: This number needs to be the same blocking factor that was used for the backup.

`-f` *device* | *file* | `-`	Specify the file name or device from which the backup data is read. For example:
	`-f ./backup.0515`
	`-f /dev/rmt/0`

`-f -`	This option reads from standard in (`stdin`) to pipe the restore from another tool (not possible in interactive mode).
`-i`	Interactive mode (see the following section on interactive mode).
`-t 1 | 2 | 3`	Designates the type of restore to be performed: • 1=regular restore (default)–To help a user who accidentally expunged a folder to recover messages. The *username* should be specified on the command line. If the original folder exists, folders are restored using their names, followed by the date of the backup. For example, the folders will be in the form: *mailbox.date*. If *mailbox.date* already exists, `imrestore` uses *mailbox.date*.`001`, then *mailbox.date*.`002`, and so on. • 2=recover–To be used when the message store has been corrupted and was reinitialized. A recover should be done on the last full backup, plus every incremental backup since that full backup, in chronological order. Folders are restored to their original names whenever possible. • 3=migrate–Used to migrate users from one message store to another. The *username* needs to be specified on the command line.

`-c` *continuation_filename*	Used when restoring from multiple backups *that were made from the same message-store*. Allows the restored messages' identifiers to be loaded from and saved to the specified file so that subsequent restores from continuation backups don't create multiple copies of identical messages. When this option is not used, an in-memory table is created for the lifetime of this restore session. This option should only be used for backups made on the same message-store when no full restore has occurred between backups. If in doubt, *do not* use this option.
`-u` *usernames_file* | *userlist*	Used only to restore or migrate users whose names are in the *usernames_file* or specified users (*userlist*). By default, all users on the back up tape are restored or migrated.
`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Interactive Mode

`imrestore -i` works with an interface that accepts the following commands:

`B` *blkfactor*	Set blocking factor (default: is 20)
`L` [*device* | *file*]	Open catalog or list files on device (this is the default)
`O` [*usernum* | *name*]	Open *usernumber* | *username*'s backup file (if no argument is given, the current user is displayed)
`C`	Close current user file.
`F` [*foldername*]	View contents of *foldername* in current file
`M` [*folder* | *num*]	View message number (*num*) from folder (*foldername*).
`T 1 | 2 | 3`	Set restore options to: • 1 - Regular restore (undelete) • 2 - Recover (crash recovery) • 3 - Migrate users
`RM` [*folder*] *num*	Restore message number, *num,* from *folder* name
`RF` [*folder*]	Restore *folder*

`RU [ `*newusername*]	Restore user to *newusername*
`RALL`	The `RALL` command restores all users in the catalog according to the current type set by `T`.
`?`	Help

The first thing to do in interactive mode is to open the back up device by using the `L` command with an argument, if none was specified with `-f`. This displays the list of users who were backed up on this tape, along with information about the actual back up date.

For each user you want to restore, that user file must be opened with the command `O` [*username* | *name*]. The list of folders that exist for this user is displayed. It is now possible to:

- View the contents of a folder with the `F` command (this displays the header of each message present).
- View a particular message in a folder with the `M` command.
- Set the type of restore to `1`, `2`, or `3` (using the `T` option as defined in the preceding table).
- Restore or migrate a single message, folder, or the entire user information using the commands `RM`, `RF`, and `RU`, respectively.

imsasm

`imsasm` is an external Solstice Backup ASM (Application Specific Module) that handles the saving and recovering of user mailboxes. `imsasm` is used in Solstice Backup (Networker) and invokes the `imbackup` and `imrestore` utilities to create and interpret a data stream.

During a save operation `imsasm` creates a save record for each mailbox or folder in its argument list. The data associated with each file or directory is generated by running the `imbackup` or `imrestore` command on the user's mailbox.

When browsing the file details with the `nwrecover` program, files (mailboxes) saved with `imsasm` will appear empty, but the full contents will be restored when they are actually recovered.

Syntax

```
imsasm [standard-asm-arguments]
```

See `uasm(8)` for a general description of ASM and the *standard-asm-arguments*.

Example

To use `imsasm` to save the mailbox `INBOX` for user "joe', the system administrator creates a directive file `<ADM_ROOT>/backup/DEFAULT/joe/.nsr` with the following contents:

```
imsasm: INBOX
```

This causes the mailbox to be saved using `imsasm`.

Note – `mkbackupdir` will generate the `.nsf` files automatically. See "`mkbackupdir`" on page 29 for more information on the `mkbackupdir` utility.

imsinit

`imsinit` is the utility that initializes the message store file system.

The top-level directories are specified in the `/etc/opt/SUNWmail/ims/ims.cnf` file. If a default SIMS installation has been performed, these directories are:

- `/var/opt/SUNWmail/ims/index`
- `/var/opt/SUNWmail/ims/hash`
- `/var/opt/SUNWmail/ims/data`
- `/var/opt/SUNWmail/ims/adm`
- `/var/opt/SUNWmail/ims/shared`
- `/var/opt/SUNWmail/ims/user`

The preceding directories must also be owned by the message store owner as specified in the `ims.cnf` file. If a default SIMS installation has been performed, the owner should be set to inetmail.

If the top-level directories are not present `imsinit` will create them.

Upon successful completion, the message store filesystem is initialized.

Syntax

```
imsinit [-l config_file_location]
```

The option for this command is:

`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

`mkbackupdir`

The `mkbackupdir` utility creates and synchronizes the backup directory with the information in the message store. It is used in Solstice Backup (Legato Networker). The `backup` directory is an image of the message store. It does not contain the actual data. `mkbackupdir` scans the message store's user directory, compares it with the `backup` directory, and updates the `backup` directory with the user names and mailbox names under the message store's user directory.

The `backup` directory is created to contain the information necessary for Networker to backup the message store at different levels (server, group, user, and mailbox). FIGURE 2-1 displays the structure.

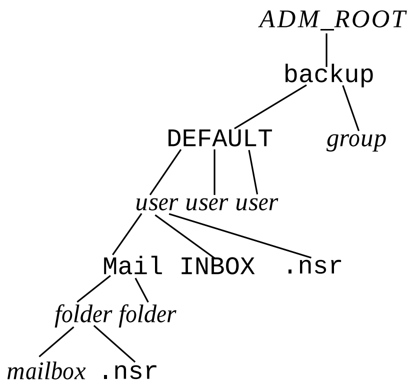

FIGURE 2-1 Backup directory hierarchy

The variables in the backup directory contents are:

ADM_ROOT	The message store administrator root directory as specified in the `/etc/opt/SUNWmail/ims/ims.cnf` file. The default directory is `/var/opt/SUNWmail/ims/adm`.
group	The user-defined group directory created by the system administrator.
user	Name of the message store user.
folder	Name of the user mailbox directory.
mailbox	Name of the user mailbox.

The `mkbackupdir` utility creates:

- a *user* directory under the `backup` directory for each new user in the message store
- a user folders hierarchy under the *user*/`Mail` subdirectory
- a `.nsr` file for each subdirectory that contains user mailboxes

The user folder hierarchy (*user/*`Mail`) contains the same structure as the *user*/`Mail` directory in the message store. The `INBOX` and the user mailboxes under the folder hierarchy contain zero length files that represent the mailbox names that are to be saved. They do not contain the actual data.

The `.nsr` file is the NSR configuration file that informs the Networker to invoke `imsasm`. `imsasm` then creates and interprets the data stream.

Each user mailbox is a file of zero length. This includes the `INBOX`, which is located under the *user* directory.

Syntax

```
mkbackupdir [-d date] [-l config_file_location]
```

The options for this command is:

-d *date*	Creates a backup directory that contains only those mailboxes that have been modified since the specified date. *date* must be of the form *yyyymmdd*, where *yyyy* specifies the year, *mm* specifies the month, and *dd* specifies the day. This option is used to perform incremental backups.
`-l` *config_file_location*	Location of Message Access Services configuration file (`ims.cnf`). If the `-l` option is not specified, the default location (`/etc/opt/SUNWmail/ims/`) will be used.

Example

To create the *ADM_ROOT/backup* directory, enter the following:

```
% mkbackupdir
```

Sun Directory Services

This section describes the Lightweight Directory Access Protocol (LDAP) utilities. These commands are in the `/opt/SUNWconn/ldap/bin` and `/opt/SUNWconn/ldap/sbin` directories. The SIMS directory service stores naming and addressing information for the components of SIMS, and profiles for the users of those components. For example:

- The IMTA uses the directory to store email user profiles and distribution lists.
- The Legacy Services Hub uses the directory to synchronize information held in various proprietary directories and messaging systems.
- The Internet Message Access Server uses the directory to verify the credentials of users and to store user message store configuration information.

The following command line utilities are used for directory usage and are outlined in this section. Detailed information about these utilities can be found in their individual man pages.

Specifying a Distinguished Name

A distinguished name (DN) is a sequence of relative distinguished names (RDNs), separated by a comma, for example `o=XYZ, c=US`. To specify a DN, refer to "`ldapadd`," "`ldapmodrdn`," and "`ldapmodify`."

Regular Expressions

You can specify a set of entries using a regular expression. See the `regex(1F)` man page for information about regular expressions.

You can specify a regular expression for the distinguished name of an entry. For example, the regular expression `dn="cn=Joe Smith, ou=.*, o=XYZ, c=US"` specifies the set of entries for people called Joe Smith in the whole of the XYZ Corporation. DN-based regular expressions are useful when defining access controls.

You can also use a DN-based regular expression to specify a set of values for an attribute whose values are DNs. For example, you can grant write access to a distribution list entry to any person whose DN is a value of the member attribute, using the regular expression `member="dn=.*"`.

Specifying an LDAP Filter

A (LDAP) filter is a way of specifying a set of entries, based on the presence of a particular attribute or attribute value. You can use an LDAP filter in access control rules. For example, the default access control rules include a filter specifying that users can add their own DNs to the member attribute of any entry that contains the joinable attribute with a value of TRUE. This allows users to add or remove their names from distribution lists.

Refer to the section entitled "`ldapsearch`." Additional information about LDAP directory concepts and configurations are included in Chapter 4, "Sun Directory Services Configuration," in the section entitled "LDAP Directory Concepts."

idxgen

The `idxgen` command regenerates the index files for the data store whose suffix you specify. If you do not specify a suffix, `idxgen` regenerates all indexes for all data stores on the machine where you execute the command.

Syntax

```
idxgen [ datastore ]
```

Example

The parameter for this command is:

datastore	Suffix of the data store for which you want to regenerate the index.

```
% idxgen datastore
```

imldifsync

The `imldifsync` command synchronizes LDAP directory entries with data in `passwd` format and data in `aliases` format. It is used to generate and update directory entries for users and for groups in LDAP Directory Interchange Format (LDIF). The LDIF file format is described in `ldif`(4) and `slapd.replog`. Entries created from the content of the LDIF file can be added to an LDAP directory using `ldapadd` or `ldapmodify`.

The `imldifsync` command runs in two modes that are mutually exclusive: user mode (option `-u`) to create user entries, and group mode (option `-g`) to create group entries. When you create or update your directory database, you need to run `imldifsync` twice: first in `user` mode, then in `group` mode. It is important to generate users first and apply the changes to the directory database before generating groups.

To generate user entries and email addresses, the `imldifsync` command uses the *password* file and *alias* file. The common name of each user entry is generated from the `gecos` field (the fifth field in the password file) by a conversion script. You can specify your own conversion script using the `-G` option if the default conversion does not meet your requirements. For details, see “`gecos2cn_script`”.

To generate group entries, the `imldifsync` command uses primarily the *alias* file. Information about the members of a group is taken from the directory database, from the previously generated user entries.

Each entry must have a unique name. If two entries have the same name, the second entry is written to a temporary file in `/tmp` and a warning message is generated. Entries for which a proper common name cannot be created are ignored, and an error is generated.

When the program exits (or is terminated by CTRL-C), it prints some statistics to `stderr` indicating how many DNs were added, modified, or found to be duplicates. In the case of duplicates, it indicates the name of the temporary file to which they were written.

Syntax

You can specify the options for `imldifsync` directly on the command line, or within a configuration file, or a combination of both methods. Each command line option has a corresponding keyword in the configuration file. The format of the configuration file is described in "configfile."

When options are specified on the command line, the syntax for `imldifsync` is as follows:

```
imldifsync [-a addvalsfile] [ -A aliasesfile [ -A aliasesfile ... ] ] [ -B grpbasedn ]\
[ -c configfile ] [ -d debuglevel ] [ -G gecos2cn_script ] [ -h ldaphost] [ -H mailserver ]\
-i [ -m maildomain ] [ -M mailsuperdomain ]\
[-O generate={ SIMSmail | IMCalendar}] [ -p ldapport] }\
[ -P passwdfile ] [ -r ] [ -S shadowfile ] [ -v SIMSversion ] [ -x excludeattrfile ] \
[ -X excludeDNfile ] [ -b basedn ] [ -D binddn ] -u | -g [ -w passwd ]
```

When options are specified in a configuration file, the syntax for `imldifsync` is as follows:

```
imldifsync [ -c configfile ] [-b basedn] [-D binddn] [ -u | -g ] [-w passwd ]
```

When options are specified both on the command line and in the configuration file, the command line options override the configuration file options. The exceptions to this rule are:

- Option `-A` (alias file)–Where the files specified in *configfile* are added to the list of alias files
- Options `-u` and `-g`–Where you must specify the mode in *configfile* or on the command line, but not both

The keywords to use to specify options in the configuration file are listed in the table in the following section "configfile."

The parameters for this command are:

[`-a` *addvalsfile*]	Specifies that the LDIF information contained in *addvalsfile* must be appended to every entry generated by `imldifsync`. Refer to "addvalsfile" for details. You can specify an *addvalsfile*, in the *configfile* specified using option `-c`, as follows: `add-vals-file = "`*addvalsfile*`"` Values placed in the *addvalsfile* may only be chosen from the SIMS schema; user generated object classes and / or attributes are not supported in the *addvalsfile*.

[`-A` *aliasesfile*]	Specifies an *aliasesfile* that contains data in `aliases`(4) format. The default for *aliasesfile* is `/etc/mail/aliases`. The information in *aliasesfile* is used to generate group entries and mailbox entries for users. An alias for which there is an owner-alias is treated as a group. An alias for which there is no owner-alias is treated as a mailbox for an individual user. (This behavior can be modified with the `-i` option).
	
	When generating user entries, `imldifsync` searches *aliasesfile* for aliases for a *username*. The set of aliases defined for a *username* becomes the set of `rfc822Mailbox` values for that user's directory entry. Of these `rfc822Mailbox` values, one is the `preferredRfc822Originator` and one (possibly a different one) is the `preferredRfc822Recipient`.
	
	When generating group entries, the *aliasname* and the `address` fields are used to generate values for the common name, member and owner attributes. The name of the alias is used to create the RDN of the directory entry. The names of aliases in the *aliasesfile* must be unique.
	
	In `sendmail`, an alias cannot contain more than about 1000 characters. `imldifsync` does not have such a limit. A mail alias file can include other alias files specified using `:include:` lines, as described in `aliases`(4). Multiple alias files can be specified by repeating the -A option. This is useful if your `sendmail.cf` file specifies several alias files. Alternatively, you can specify a list of alias files in the *configfile* specified using option `-c`, as follows:
	
	`alias-file = "`*filename*`"`
	`alias-file = { "`*filename*`" , "`*filename*`" ... }`

[`-b` *basedn*]	Specifies the Distinguished Name (DN) of a base entry in the directory. New user entries are named relative to this base entry. *basedn* must be a string-represented DN. If the *basedn* specified does not exist, the LDIF information necessary to create it is generated automatically. The `-b` option is required in user mode and in group mode. You can specify a *basedn* in the *configfile* specified using option `-c`, as follows: `base =` “*basedn*”

`-B` *grpbasedn*	Specifies the Distinguished Name (DN) of a base entry when `imldifsync` is used in group mode. New group entries are named relative to this base entry. *grpbasedn* must be a string-represented DN.
	
	If you do not specify a *grpbasedn*, the *basedn* specified using the `-b` option is used. The `-B` option is useful if you want to store group and user entries in different parts of the directory tree. For example, to store user entries under ou=people, `ou=sales`, `o=XYZ`, `c=US` and group entries under `ou=groups`, `ou=sales`, `o=XYZ`, `c=US`, you would run `imldifsync` in group mode (`-g`), with the following options:
	
	`-b ou=people,ou=sales,o=XYZ,c=US`
	`-B ou=groups,ou=sales,o=XYZ,c=US`
	
	You can specify a *grpbasedn* in the *configfile* specified using option `-c`, as follows:
	
	`group-base = "`*grpbasedn*`"`

`-c` *configfile*	Specifies the name of a configuration file that can contain values for all the other options of the `imldifsync` command. For details on the format of *configfile*, refer to “configfile.”
`-d` *debuglevel*	Displays debugging information on standard error. *debuglevel* is a number from 0 to 3. You can specify a *debuglevel* in the *configfile* specified using option `-c`, as follows: `debug =` *debuglevel*

`-G` *gecos2cn_script*	Specifies the name of the program or script to use to convert information in the `gecos` field of the *password* file to a common name for an entry. The `-G` option can only be used when running `imldifsync` in user mode (`-u` option). Use `-G` if the `gecos` field (the fifth field) in your *password* file is not formatted in a way that is suitable for use with the default `gecos` conversion function. The *gecos2cn_script* program is started by `imldifsync` and is used many times when user information is being generated. Your *gecos2cn_script* runs with the same userid as `imldifsync`. If you run `imldifsync` in the crontab, both programs run as root. "`gecos2cn_script`" provides details of how to create a *gecos2cn_script* program, and explains how the default conversion program works. You can specify a *gecos2cn_script* in the *configfile* specified using option `-c`, as follows: `imgecos2cn-prog = "`*gecos2cn_script*`"`
`-h` *ldaphost*	Specifies the host on which the `slapd` server daemon is running. The default is *localhost*. You can specify an *ldaphost* in the *configfile* specified using option `-c`, as follows: `ldap-host = "`*ldaphost*`"`
`-H` *mailserver*	Specifies the *mailserver* for all users. By default, when creating a directory entry for a user, if that user has a mail alias containing an @ sign, the hostname on the right hand side of the @ is used as the *mailserver.* If no user alias contains the mailserver, and no value is specified using `-H`, the hostname which is aliased to mailhost in the hosts database is used as the mailserver. You can specify a *mailserver* in the *configfile* specified using option `-c`, as follows: `mail-server = "`*mailserver*`"`

`-i`	The `-i` option is *required* when using `imldifsync`. Used in conjunction with `-g` (group mode), the `-i` option indicates to `imldifsync` that it should generate a group entry for every mail alias in the aliases file. The default is to generate a group entry only for an alias that has an owner defined in the mail aliases file. If you specify the `-i` option, for an alias that does not have an owner, the postmaster's distinguished name is used. If no distinguished name is defined for the postmaster, the DN for root is used and a warning message is displayed, since root does not necessarily have an entry in the directory. Used in conjunction with `-u` (user mode), the `-i` option indicates to `imldifsync` to generate user entries for all users. The default is to generate entries only for users that have an alias defined in an *alias* file. You can specify this behavior in the *configfile* specified using option `-c`, as follows: `include-all = true`
`-m` *maildomain*	Specifies the mail domain for all users. If specified, this string is appended to any *hostname* that does not contain a dot or to hosts that are in the local domain (that is, hostnames for which `gethostbyname`() does not fail). This field is appended to hostnames such as users' mail server. If neither the `-m` option, nor the `-M` option is specified, the default *maildomain* is the domain name of the host from the "service ready" message on the SMTP port of the mailserver (see `-H` option to set the mailserver). You can specify a *maildomain* in the *configfile* specified using option `-c`, as follows: `mail-domain = "`*maildomain*`"`

`-M` *mailsuperdomain*	Specifies the *superdomain* of the network. This string specifies the domain that is appended to domain names. It is appended to domain names to fully qualify the hostname. For example, if a user is listed in a group as `joe@ireland` and `ireland` is not a host (that is, `gethostbyname`( ) for `ireland` fails), `ireland` is assumed to be a domain and *mailsuperdomain* is appended to it. If neither the `-m` nor the `-M` option is specified, the default for *mailsuperdomain* is the *superdomain* of the *maildomain* (see the `-m` option). That is, everything after the first dot. For example, if the *maildomain* is `Boston.XYZ.COM`, the default *superdomain* is `XYZ.COM`. If you do not want *superdomains* appended to domains, specify `-M ""`. This is useful in an internal network where the *superdomain* is never appended to *hostnames*. You can specify a *mailsuperdomain* in the *configfile* specified using option `-c`, as follows: `super-domain = "`*mailsuperdomain*`"`

`-O generate=SIMSmail`
`-O generate=IMCalendar`
	For the user phase only, specify whether Directory objects for SIMS email user or the calendaring part of WebAccess should be generated. The `generate`=*option_name* syntax may be a concatenated, comma-separated argument to `-O`, or you may have multiple `-O` arguments on the command line. `SIMSmail` specifies that the objects for SIMS email support are generated. `IMCalendar` specifies that calendar objects for the WebAccess calendar client is created. If none of these are specified, `SIMSmail` is assumed
	
	You may specify `generate` directive in the *configfile* specified using the `-c` option, as follows:
	`option`={“`generate=SIMSmail`”,
	“`generate=IMCalendar`”}

`-p` *ldapport*	Specifies an alternate TCP port where the `slapd` server is listening. The default port is 389. You can specify an *ldapport* in the *configfile* specified using option `-c`, as follows: `ldap-port = "`*ldapport*`"`

`-P` *passwdfile*	Specifies a file that contains data in `passwd`(4) format. By default, the *passwd* entry from `/etc/nsswitch.conf` is used. If your `/etc/nsswitch.conf` is set up to read from more than one source, it is possible that both sources could have different information for the same entry, causing the output from `imldifsync` to be inconsistent. In this case, use a utility like `ypcat` or `niscat` to create a file that you can specify, using the `-P` option. If a password file is not specified, `imldifsync` uses `getpwent`(3C) and `getspent`(3C), which get the user information from the location specified in `/etc/nsswitch.conf`. You can specify a *passwdfile* in the *configfile* specified using option `-c`, as follows: `passwd-file = `"*passwdfile*"
`-r`	This option controls whether an entry for a user or group that is not found in the input files is to be deleted from the directory. If you supply this option, when a user or a group no longer exists, `imldifsync` generates delete operations to remove the old entries from the directory database. You can specify this behavior in the *configfile* specified using option `-c`, as follows: `remove-unfound-entries = true`
`-S` *shadowfile*	If a *passwdfile* is specified using `-P`, a separate shadow file can be specified using `-S`. If neither the `-P` option nor the `-S` option is specified, `imldifsync` uses `getpwent`(3C) and `getspent`(3C), which get the user information from the location specified in `/etc/nsswitch.conf`. You can specify a *shadowfile* in the *configfile* specified using option `-c`, as follows: `shadow-file = `"*shadowfile*"
`-v` *SIMSversion*	The `-v` option specifies the version of Sun Internet Mail Server(SIMS). Indicates to `imldifsync` to generate `LDIF` appropriate for a particular version of SIMS. Currently, the only version supported is `3.5`. Version 3.5 is the default if this option is not specified. You can specify the *SIMSversion* in the *configfile* specified using option `-c`, as follows: `version= `"*SIMSversion*"

`-x` *excludeattrfile*	The attributes names contained in *excludeattrfile* are ignored when reading and updating entries in the directory. Place each attribute on a separate line with no leading or trailing white space. This is useful if you add attributes to each entry by hand and you don't want `imldifsync` to modify or delete these attributes subsequently. Protecting an attribute in this way does not prevent an entire entry that includes that attribute from being deleted.
	
	You can indicate to `imldifsync` both to ignore an attribute (with the `-x` option) and to update it by adding it to each entry (in the *addvals* file with the `-a` option or in the *gecos2cn_script*). In this case, the value is added to new entries but not modified in existing entries. You can specify an *excludeattrfile*, in the *configfile* specified using option `-c`, as follows:
	
	`add-vals-file = "`*excludeattrfile*`"`
	
	You can also specify attributes to be ignored by putting them directly into the *configfile*. A single attribute can be specified as follows:
	
	`ignore-attr = "`*attr-to-ignore*`"`
	
	You can specify a list of attributes to be ignored as follows: `ignore-attrs = { "`*attr-to-ignore*`", "`*attr-to-ignore*`", ...}`
	
	You can specify any number of *ignore-attr* and *ignore-attrs* lines in *configfile*.

`-X` *excludeDNfile*	The DNs contained in *excludeDNfile* specify directory entries that are excluded from processing. The DN must match exactly the DN in the directory database. If an entry's DN changes to a DN that is in the exclude file, that entry is not modified; it is excluded from further processing, and a warning message is generated. You can specify an *excludeDNfile*, in the *configfile* specified using option `-c`, as follows:
	
	`add-vals-file = "`*excludeDNfile*`"`
	
	You can also specify the DN of an entry to be excluded by putting it directly in the *configfile.* A single entry can be specified as follows:
	`excludeDN = "`*dn-to-exclude*`"`
	
	Multiple entries can be specified as follows:
	
	`excludeDNs = { "`*dn-to-exclude*`", "`*dn-to-exclude*`", ...`
	`}`
	
	You can specify any number of *dn-to-exclude* lines in the *configfile.*

`-D` *binddn*	Use the distinguished name *binddn* to bind to the `slapd` server. You can specify a *binddn* in the *configfile* specified using option `-c`, as follows: `rootdn= "`*binddn*`"`
`-u`	Generates user directory entries from the password file. You must specify either the `-u` option or the `-g` option. An entry is created for every user in the password file who also has an alias in the mail aliases file. This prevents user entries from being created for noninteractive users such as `nobody` or `lp`. The attributes and values created by `imldifsync` for each entry are described in "Output–User Mode." Other attributes can be added by returning them in a user defined *gecos2cn_script* (`-G` option) or by putting them in an *addvals* file (`-a` option). You can specify the behavior of `imldifsync` in the *configfile* specified using option `-c`, as follows: `mode = users`

`-g`	Generate group directory entries from the mail aliases file. You must specify either the `-g` option or the `-u` option. For an alias in the mail aliases file to be considered a group, an owner alias must also be defined in the aliases file (unless the `-i` option is specified). Within the owner alias, there must be at least one user with an entry in the directory. If there is no owner who has a directory entry, the DN for postmaster is used. If there is no directory entry for postmaster, the DN for root is used and a warning message is displayed, since root does not necessarily have an entry in the directory. The attributes and values created by `imldifsync` for each entry are described in “Output Format.” Other attributes can be added by returning them in a user-defined *gecos2cn_script* (`-G` option) or by putting them in an *addvals* file (`-a` option). `imldifsync` requires directory information about users when creating the group definitions. You must create or update the users with the `-u` option, before you create or update the groups. You can specify the behavior of `imldifsync` in the *configfile* specified using option `-c`, as follows: `mode = groups`
`-w` *passwd*	Use *passwd* as the password to bind to the `slapd` server. You can specify a *passwd* in the *configfile* specified using option `-c`, as follows: `ldap-passwd=` “*passwd*”

Format of Input Files

This section describes the format of files that can be specified as input to the `imldifsync` command:

- *addvalsfile*
- *configfile*
- *gecos2cn*

addvalsfile

To replace values, lines in *addvalsfile* must be in the format:

```
attrname: attrvalue
```

To add values, lines in *addvalsfile* must be in the format:

```
attrname+ attrvalue
```

Specify each attribute/value pair on a separate line. *addvalsfile* is read and the attributes in each entry are compared with each attribute specified. Any entry that contains this attribute is updated. Existing values for the named attribute are replaced with the values specified with a colon (:), and values specified with a plus sign (+) are added to the attribute. All the attributes and values in *addvalsfile* are added to new entries.

You can also specify additional values to be added by putting them directly into the *configfile* specified using the `-c` option. A single value can be specified by adding the line:

```
addval = "attrname: attrvalue"
```

or

```
addval = "attrname+ attrvalue"
```

Multiple attributes can be specified as follows:

```
addvals = { "attrname: attrvalue", "attrname+ attrvalue", ... }
```

Multiple instances of each form can be used, with a mixture of : and + terms. Values placed in the `addval` directive may only be chosen from the SIMS schema; user generated objectclasses and/or attributes are not supported in the `addvalsfile` or the `addval` directive in the configfile.

configfile

When you specify options in the *configfile*, they must have the general form:

`option =` *value*

If *value* is a string, the string must be in double quotes. If *value* is a switch, it can be enabled with the word `TRUE` or `1` and disabled with the word `FALSE` or `0` (without quotes). If *value* is a number, the number is specified without quotes. If *value* is a list, it is specified as {comma separated list}. Keywords such as `TRUE`, `FALSE`, users, and groups are specified without quotes.

If options are also set on the command line, they override the options in *configfile*. The exceptions to this rule are:

- Option `-A` (alias file), where the files specified in *configfile* are added to the list of alias files
- Options `-u` and `-g`, where you must specify the mode in *configfile* or on the command line, but not both.

The following table gives the syntax for specifying `imldifsync` options in the *configfile*.

Option	Keyword

`-a` *addvalsfile*	`add-vals-file = "`*addvalsfile*`"`
	`add-val = "`*attrname*: *attrvalue*`"`
	`add-val = "`*attrname+ attrvalue*`"`
	`add-val = { "`*attrname*: *attrvalue*`","`*attrname+ attrvalue*`", ... }`

`-A` *aliasesfile*	`aliases-file = "`*filename*`"`
	`aliases-file = { "`*filename*`" , "`*filename*`" ... }`

`-b` *basedn*	`base = "`*basedn*`"`
`-c` *configfile*	`config-file` = "*configfile*"
`-B` *grpbasedn*	`group-base = "`*grpbasedn*`"`
`-d` *debuglevel*	`debug =` *debuglevel*
`-G` *gecos2cn_script*	`gecos2cn-prog = "`*gecos2cn_script*`"`
`-h` *ldaphost*	`ldap-host = "`*ldaphost*`"`
`-H` *mailserver*	`mail-server = "`*mailserver*`"`
`-i`	`include-all = TRUE`
`-m` *maildomain*	`mail-domain = "`*maildomain*`"`
`-M` *mailsuperdomain*	`super-domain = "`*mailsuperdomain*`"`
`-O`	`generate = SIMSmail, IMCalendar`
`-p` *ldapport*	`ldap-port = "`*ldapport*`"`
`-P` *passwdfile*	`passwd-file = "`*passwdfile*`"`
`-r`	`remove-unfound-entries = TRUE`
`-S` *shadowfile*	`shadow-file = "`*shadowfile*`"`
`-v` *SIMSversion*	`version= "`*SIMSversion*`"`

`-x` *excludeattrfile*	`ignore-attr-file = "`*excludeattrfile*`"`
	`ignore-attr = "`*attr-to-ignore*`"`
	`ignore-attr = { "`*attr-to-ignore*`", "`*attr-to-ignore*`", ...}`

`-X` *excludeDNfile*	`exclude-dn-file = "`*excludeDNfile*`"`
	`exclude-dn = "`*dn-to-exclude*`"`
	`exclude-dn = { "`*dn-to-exclude*`", "`*dn-to-exclude*`", ... }`

`-D` *binddn*	`bind-dn= "`*binddn*`"`
`-u`	`mode = users`

`-g`	`mode = groups`
`-w` *passwd*	`ldap-passwd = "`*passwd*`"`
`-t` *ldap-timeout*	`ldap-timeout` = *seconds*

The file `/etc/opt/SUNWmail/dir_svc/imldifsync.conf` is a sample *configfile*.

gecos2cn_script

`gecos2cn_script` is the conversion program that `imldifsync` uses to generate user entries in LDIF format from the information found in the `passwd` file. The following information is extracted from the `passwd` file and is input to the conversion program:

```
username gecos-field
```

The `username` is the first word on each line; the *gecos-field* is all the remaining words on the line.

Default Conversion

If you do not specify your own conversion program with the option `-G`, `imldifsync` expects the `gecos` field to be in the following format:

```
given-names  surname,  generation qualifier  -  comment
```

`imldifsync` interprets the `gecos` field according to this pattern, and applies the following rules:

1. If there is a comma in the `gecos` field, the words that follow the comma are the generation qualifier, the word that precedes it is the surname.
2. Anything following a space-dash-space sequence is a comment (for example job title or nickname).
3. The surname can be composed of several words: imldifsync interprets words that are all lower case as being part of the surname (for example the naming prepositions “van der” in German or “de” in French, are part of the surname). It also interprets words that are in all capitals as being part of the surname.
4. Anything enclosed in double quotes or between brackets (even a nonmatching pair such as this] is interpreted as a comment, whatever the position in the `gecos` field.

Given names, surname, and generation qualifier must start with an alphabetic character, can contain alphabetic characters, dashes (-), and single quotes (‘), and must end with an alphabetic character or a period. For example, “Joe,” “John-Paul,” “O’Connor,” or “Jr.” are permitted, but not “John+” or “-Sam.”

The generation qualifier is optional, but, if present, must not be blank. For example, you cannot have a `gecos` field such as “Joe Cool, - expert” because of the blank space after the comma. If there is only one word in the `gecos` field, it is assumed to be the surname. If there is none, the username is assumed to be the surname.

If the preceding format for the `gecos` field is *not* observed, LDIF directory entries are still generated, and warning messages are displayed. `imldifsync` creates the following attributes for each directory entry:

- `cn` (common name)
- `givenName`
- `initials`
- `sn` (surname)
- `generationQualifier`

For example, if the username is “john” and the `gecos` field in the `passwd` file is:

```
John-Pierre Humphrey (Buster) Smith, Jr. - Technical Support
```

By default, `imldifsync` creates the CN and other attributes as follows:

```
cn:"John-Pierre H. Smith, Jr. (john)"
cn:"John-Pierre H. Smith, Jr."
cn:"John-Pierre Humphrey (Buster) Smith, Jr. - Technical Support"
givenName:John-Pierre
initials:JPHS
sn:Smith
generationQualifier:Jr.
```

Custom Conversion Program

If the format of the `gecos` field in your `passwd` file does not follow the format outlined in “Default Conversion” `imldifsync` is unlikely to produce usable results. If this is the case, you can specify your own conversion program using the option `-G`. Your program must adhere to the following rules.

Your program must return a set of `attribute-value` pairs for each user entry, with one `attribute-value` pair per line, and a blank line between each set. The format for specifying an `attribute-value` pair is:

```
attributename:value
```

or

```
attributename+value
```

Your program can return a value for any attribute defined in the schema. It must return at least one value for the `cn` (common name) attribute and one value for the `sn` (surname) attribute. The first value returned for `cn` is used as the RDN of the entry.

Your program can return several values for an attribute that the schema defines to be multivalued, using `:` notation for values that replace existing values and `+` notation for values to be added. Do not use the `+` notation for the attributes `cn`, `sn`, `initials`, `givenName`, or `generationQualifier`. Return each value on a separate line with the same attributename. Multivalued attributes must be grouped together on consecutive lines.

If an error occurs when you are parsing the `gecos` field, your program must return `FAIL=` followed by an error message. No space before or after the `=` is expected. The `FAIL=` line ends the entry. Do not supply a blank line after a `FAIL=` line. If parsing of the `gecos` field fails, the error message after the `FAIL=` is printed by `imldifsync` and the processing stops for that entry.

The following sample shows the information communicated between `imldifsync` and a `gecos2cn_script` The lines beginning with “-->” show information sent to the script and the lines beginning with “<--” show information received from the script. (The “-->” and “<--” characters are not part of the output.)

```
--> jp John-Paul Humphrey (Buster) Smith, Jr. - Technical Support
<-- cn:SMITH, John-Paul Humphrey (Buster), Jr.
<-- sn:SMITH
<-- givenName=John-Paul Humphrey (Buster)
<-- initials:JHS
<-- generationQualifier:Jr.
<--
--> bad @#$!@#$@#$ %$# this is a bad gecos field
<-- FAIL= error parsing gecos field
--> sue Susan Jones
<-- cn:JONES, Susan
<-- sn:JONES
<-- givenName:Susan
<-- initials:SJ
<--
```

In the preceding example, the script is given an entry for `jp`. It returns five lines for his entry. Then the script is given the entry for bad, and responds with an error. Then the script is given the entry for `sue` and returns four lines of output for her entry.

A sample script is provided in `/opt/SUNWmail/dir_svc/samples/imgecos2cn.sh`. This script is similar, but not identical, to the default conversion function of `imldifsync`.

Output–User Mode

Used in user mode with the `-u` option, the `imldifsync` command creates entries with the following attributes and values:

`cn, sn, initials, givenName,` and `generationalQualifier`	These fields are created either by the built-in conversion function or by the `gecos2cn` program specified with the `-G` option. See the `-G` option for details on both.
`mailDeliveryOption`	This is a multivalued attribute describing how mail addressed to this user is delivered: `mailbox`: The message is delivered to message store for this user. • `native`: The message is appended to `/var/mail/userid`. • `program`: The message is delivered to every program named in the `mailProgramDeliveryInfo` attribute. • `forward`: The message is delivered to every user listed in the `mailForwardingAddress` attribute. • `file`: The message is appended to every file named in the `mailDeliveryFile` attribute.
`rfc822Mailbox`	The values of this attribute are generated by searching for mail aliases that map exactly to this user, that is, aliases with only one name on the right side. This set of attribute values is a list of all the user's possible addresses, not a list of the mail aliases to which the user belongs.
`preferredRfc822Originator`	The value of this attribute is the return address specified for the user on outgoing mail. It is the fully qualified alias name of the first alias in the set of values specified for the rfc822Mailbox attribute that contains an @ in the name on the right side of the alias. For example, if the mail alias is defined as Mary.Smith: msmith@mailserver, the value of preferredRfc822Originator is Mary.Smith@mailserver.XYZ.com.
`preferredRfc822Recipient`	This field is where the mailer routes the user's incoming mail. It is the first alias on the user's chain of aliases that contains an @ in the name on the right side of the alias, for example, the right side of an alias like `Mary.Smith: Mary.Smith@mailserver.XYZ.com`. It is one of the mailboxes listed in the `rfc822Mailbox` attribute
`mailDeliveryFile`	If the user has a file as an alias, instead of the name appearing in `rfc822Mailbox`, the name of the file appears here.
`mailHost`	Contains the hostname on the right side of the `@` sign of the `preferredRfc822Recipient` attribute value.

mailProgramDeliveryInfo	If the user has a command as an alias, the command is a value of this attribute.
userPassword	This field is the user's password from the password file. If the password is encrypted, the password is prefixed with "{crypt}."
uid	The value of this attribute is the user's UNIX system username.
homeDirectory	The value of this attribute is the user's home directory.
dataSource	This attribute identifies the version of `imldifsync` that was used to create the entry. It is used to signal to `imldifsync` which entries it created.
objectClass	For users, the values of this attribute are `top`, `inetOrgPerson`, `organizationalPerson`, `emailPerson`, and `person`.
mailFolderMap	Preferred mail access mechanism. Two values are defined: • UNIX V7 (corresponds to a `/var/mail` message store) • Sun-MS (corresponds to a SIMS message store)

When using the `generate=IMCalendar` directive during the user mode, the following additional attributes and object classes are generated and then set:

objectClass	For users, the value `IMCalendarUser` will be added to those generated for other `generate` options.
IMcalendarHost	This attribute represents the Fully Qualified Name of the host machine which has this user's calendar server running on it. `imldifsync` initializes this value to be the same as the user's `mailHost` attribute.
IMcalendarName	This attribute will be the name of the user's calendar on `IMcalendarHost`. `imldifsync` initializes this to be a default value equal to the user's login name, as represented by the attribute `userid`.

Group Mode

Used in group mode with the `-g` option, the `imldifsync` command creates entries with the following attributes and values:

`cn`	The `commonName` is the name of the mail alias.
`mailDeliveryOption`	This is a multivalued attribute describing how a message sent to the mail aliases is delivered. Permitted values are: • `mailbox`: The message is delivered to every recipient named in the member attribute and to every recipient named in the `rfc822MailMember` attribute. • `program`: The message delivered to every program named in the mailProgramDeliveryInfo attribute. • `file`: The message is appended to every file named in the `mailDeliveryFile` attribute.
`member`	Each value of the member attribute is the distinguished name of a member of the mail alias who has an entry in the directory. A member of the mail alias who does not have an entry in the directory is listed in the `rfc822MailMember` attribute.
`rfc822MailMember`	Each value of the `rfc822MailMember` attribute is the `rfc822` email address of a member of the mail alias who does not have an entry in the directory.
`mailProgramDeliveryInfo`	Each value of the `mailProgram DeliveryInfo` attribute is a command that is a member of the mail alias.
`mailDeliveryFile`	Each value of the `mailDeliveryFile` attribute is a file that is a member of the mail alias.
`mailFolderMap`	Specifies the message store for a user. Two values are defined: • UNIX V7 (corresponds to a `/var/mail` message store) • Sun-MS (corresponds to a SIMS message store)

`ownerDeliveryOption`	This is a multivalued attribute describing how a message sent to the owner of the mail alias is delivered. Permitted values are: • `mailbox`: The message is delivered to every owner named in the owner attribute and to every recipient named in the `rfc822Owner` attribute. • `program`: The message is delivered to every program named in the `ownerProgramDeliveryInfo` attribute. • `file`: The message is appended to every file named in the ownerDeliveryFile attribute.
`owner`	Each value of the owner attribute is the distinguished name of an owner of the mail alias who has a directory entry.
`rfc822Owner`	Each value of the `rfc822Owner` attribute is the `rfc822` email address of an owner of the mail alias who does not have a directory entry.
`ownerProgramDeliveryInfo`	Each value of the `ownerProgramDeliveryInfo` attribute is a command that is an owner of the mail alias.
`ownerDeliveryFile`	Each value of the `ownerDeliveryFile` attribute is a file that owns the mail alias.
`requestsToDeliveryOption`	This is a multivalued attribute describing how a message sent to the “requests-to” alias of the mail alias is delivered. Permitted values are: • `mailbox`: The message is delivered to every recipient named in the `requestsTo` attribute and to every recipient named in the `rfc822RequestsTo` attribute. • `program` : The message is delivered to every program named in the `requestsToProgramDeliveryInfo` attribute. • `file`: The message is appended to every file named in the `requestsToDeliveryFile` attribute.
`requestsTo`	Each value of the `requestsTo` attribute is the distinguished name of a member of the `requestsTo` alias for whom the mail alias has a directory entry.
`rfc822requestsTo`	Each value of the `rfc822requestsTo` attribute is the `rfc822` email address of a member of the `requestsTo` alias for whom the mail alias does not have a directory entry.

`requestsToProgramDeliveryInfo`	Each value of the `requestsToProgramDeliveryInfo` attribute is a command that is a member of the `requestsTo` alias for the mail alias.
`requestsToDeliveryFile`	Each value of the `requestsTo DeliveryFile` attribute is a file that is a member of the `requestsTo` alias for the mail alias.
`dataSource`	This field identifies the version of `imldifsync` that was used to create the entry. It is used to signal to `imldifsync` which entries it created.
`objectClass`	For groups, this entry contains the values top, `groupOfNames`, `rfc822MailGroup`, and `emailGroup`.

Examples

The four examples in this section show how information in the password file and the mail aliases file is synchronized with the directory entries. For simplicity, access controls are ignored and the `-D` and `-w` options are not included in the `imldifsync` and `ldapmodify` commands in any of the examples.

1. To extract the aliases from NIS, execute the following command:

```
# ypcat aliases | sed 's/ /: /' > aliases.nis
```

This command produces a file that is in the same format as `/etc/mail/aliases` but is from the NIS table. If you are running `imldifsync` on the machine that is your NIS server, you do not need to do this, since `imldifsync` by default reads from the local file.

2. Type the following command to generate user directory entries. This command:
   - Parses the files `passwd.nis` and `/etc/aliases`
   - Consults the LDAP server on the localhost
   - Appends the attributes in the file named `extra` to each entry
   - Writes the generated LDIF to standard output

```
# imldifsync -u -b ou=people,o=XYZ,c=US -a extra -A aliases.nis \
-P passwd.nis -H mailserver -M XYZ.COM \
-D cn=administrator,o=XYZ,c=US -w secret
```

The file `passwd.nis` has the following content:

```
bill:7DNMzGXuqpBeU:100:10:William Smith:/home/bill:/bin/sh
bob:MRIbV9Hm6z7Wg:200:10:Robert A. Smith:/home/bob:/bin/sh
```

The file `aliases.nis` has the following content:

```
bill: william.smith
william.smith: bill@mailserver
bob: robert.smith
robert.smith: bob@mailserver
```

The file `extra` has the following content:

```
associatedDomain: XYZ.COM
channelName: smtp
messageStore: messagestore
objectClass: domainRelatedObject
```

The following LDIF is generated:

```
dn: ou=people,o=XYZ,c=US
changetype: add
cn: ou=people,o=XYZ,c=US
objectClass: top
objectClass: organizationalUnit
dn: cn="William Smith <bill>",ou=people,o=XYZ,c=US
changetype: add
cn: William Smith <bill>
cn: William Smith
sn: Smith
initials: WS
givenName: William
preferredRfc822Originator: william.smith@XYZ.COM
preferredRfc822Recipient: bill@mailserver.XYZ.COM
rfc822Mailbox: william.smith@mailserver.XYZ.COM
rfc822Mailbox: bill@mailserver.XYZ.COM
rfc822Mailbox: staff-interest-request@mailserver.XYZ.COM
rfc822Mailbox: owner-staff-interest@mailserver.XYZ.COM
rfc822Mailbox: william.smith@XYZ.COM
mailDeliveryOption: mailbox
mailHost: mailserver.XYZ.COM
userPassword: {crypt}7DNMzGXuqpBeU
uid: bill
homeDirectory: /home/bill
dataSource: imldifsync 0.10
objectClass: top
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: emailPerson
objectClass: person
objectClass: domainRelatedObject
associatedDomain: XYZ.COM
channelName: smtp
messageStore: casino
dn: cn="Robert A. Smith <bob>",ou=people,o=XYZ,c=US
changetype: add
cn: Robert A. Smith <bob>
cn: Robert A. Smith
sn: Smith
initials: RAS
givenName: Robert A.
(Cont'd)
```

```
preferredRfc822Originator: robert.smith@XYZ.COM
preferredRfc822Recipient: bob@mailserver.XYZ.COM
rfc822Mailbox: robert.smith@mailserver.XYZ.COM
rfc822Mailbox: bob@mailserver.XYZ.COM
rfc822Mailbox: robert.smith@XYZ.COM
mailDeliveryOption: mailbox
mailHost: mailserver.XYZ.COM
userPassword: {crypt}MRIbV9Hm6z7Wg
uid: bob
homeDirectory: /home/bob
dataSource: imldifsync 0.10
objectClass: top
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: emailPerson
objectClass: person
objectClass: domainRelatedObject
associatedDomain: XYZ.COM
channelName: smtp
messageStore: casino
```

Note – The `XYZ.COM`, in the `preferredRfc822Originator` and `preferredRfc822recipient` fields, was specified with the `-M` option, and was not in the `associatedDomain` added to the "extra" file.

---

3. Type the following command to generate group directory entries. This command:
   - Parses the files `/etc/mail/aliases` (the default for the `-A` option) and `passwd.nis`
   - Consults the LDAP server on the host `mars`
   - Writes the generated LDIF to standard output

```
# imldifsync -g -b ou=groups,o=XYZ,c=US -B ou=groups,o=XYZ,c=US \
-h mars -H mailserver -M XYZ.COM \
-D cn=administrator,o=XYZ,c=US -w secret
```

The file `passwd.nis` has the content specified above, and the file `/etc/mail/aliases` has the following content:

```
staff-interest: william.smith, robert.smith, joe@faraway.net
staff-interest-request: william.smith
owner-staff-interest: william.smith
```

The following LDIF is generated:

```
dn: ou=groups,o=XYZ,c=US
changetype: add
cn: ou=groups,o=XYZ,c=US
objectClass: top
objectClass: organizationalUnit

dn: cn="staff-interest",ou=groups,o=XYZ,c=US
changetype: add
cn: staff-interest
member: cn=William Smith <bill>,ou=people,o=XYZ,c=US
member: cn=Robert A. Smith <bob>,ou=people,o=XYZ,c=US
rfc822mailbox: joe@faraway.net
mailDeliveryFile: /space/staff-interest-log
mailDeliveryOption: mailbox
mailDeliveryOption: file
mailHost: faraway.net
owner: cn=William Smith,ou=people,o=XYZ,c=US
ownerDeliveryOption: mailbox
requestsTo: cn=William Smith,ou=people,o=XYZ,c=US
requestsToDeliveryOption: mailbox
dataSource: imldifsync 0.10
objectClass: top
objectClass: groupOfNames
objectClass: rfc822MailGroup
objectClass: emailGroup
```

4. Type the following command to synchronize user directory entries with data from the files `/etc/passwd` and `/etc/shadow`.

```
# imldifsync -u -b ou=people,o=XYZ,c=US -a extras -H mailserver \
-M XYZ.COM -D cn=administrator,o=XYZ,c=US -w secret | ldapmodify \
-D cn=administrator,o=XYZ,c=US -w secret
```

Only the password in the following line has changed:

`bill:aINMzmXuQp4e1:100:10:William Smith:/home/bill:/bin/sh`

The following LDIF is generated to modify the directory:

```
dn: cn=William Smith,ou=people,o=XYZ,c=US
changetype: modify
replace: userPassword
userPassword: {crypt}aINMzmXuQp4e1
-
```

The following example generates user directory entries:

```
% imldifsync -u ... -O generate=IMCalendar
```

The flags are not shown but would be present, in addition to the `-O` option (used to generate `IMCalendarUser` objects for the WebAccess suite). The example above shows generation of only `IMCalendar` objects, not including the SIMS email objects shown previously.

The following LDIF is generated:

```
dn: cn=”William Smith <bill>”,ou=people,o=XYZ,c=US
changetype: add
cn: Robert A. Smith <bob>
cn: Robert A. Smith
sn: Smith
initials: RAS
givenName: Robert A.
IMcalendarHost: faraway.Eng.Sun.COM
userPassword: {crypt}7DNMzGXuqpBeU
uid: bill
dataSource: imldifsync 1.0
IMCalendarName: bill
objectClass: top
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: IMCalendarUser
```

`ldapadd`

The `ldapadd` utility is used to add email entry tools. The entry information is read from standard input or from a file, specified using the `-f` option. The `ldapadd` command is a variant of the `ldapmodify` command. When invoked as `ldapadd`, the `-a` (add new entry) flag is turned on automatically. Additional information about modifying email entry tools can be found in the following section entitled "`ldapmodify`."

Syntax

```
ldapadd [ -bcnv ] [ -d debug_level ] [ -D binddn ] [-w my_password ] [ -h host ]\
[ -p lda_port ] [-f filename ]
```

The options for this command are:

[`-b`]	Designates attribute values in binary format.
[`-c`]	Specifies continuous operation mode. Errors are reported, but `ldapadd` continues operations.
[`-n`]	Displays what could be done, but doesn't actually modify entries. (This is especially useful when using debugging options `-d` or `-v`.)
[`-v`]	Writes detailed (verbose) diagnostic messages to the log.
[`-d` *debug_level*]	Turns on the debugging mode and specifies a debug level: • 1 - Trace • 2 - Packets • 4 - Arguments • 32 - Filters • 128 - Access control To request more than one category of debugging information, add the numbers designating the modes you want to use; for example, if you want to request trace and filter information, add 1 (for Trace) and 32 (for Filters) for a debug value of 33.
[`-D` *binddn*]	Specifies a particular Distinguished Name (DN) to bind to the directory.
[`-w` *password*]	Specifies the password to use for access to a directory.
[`-h` *ldap_hostname*]	Specifies an alternate host on which to run the `slapd` server.
[`-p` *lda_portID*]	Specifies an alternate TCP port where the `slapd` server is listening.
[`-f` *filename*]	Reads the entry modification information from a specific file instead of from standard input.

Example

The following file, `/tmp/newentry`, contains information for adding an entry:

```
dn: cn=Ann Jones, o=XYZ, c=US
objectClass: person
cn: Ann Jones
cn: Annie Jones
sn: Jones
title: Director of Research and Development
mail: ajones@londonrd.xyz.us.com
uid: ajones
```

The following command adds a new entry for Ann Jones, using the information in the file.

```
% ldapadd -f /tmp/newentry
```

`ldapdelete`

The `ldapdelete` command opens a connection to an LDAP server, binds, and deletes one or more entries. If one or more *dn* arguments are provided, entries with those distinguished names are deleted. If no *dn* arguments are provided, a list of DNs is read from *file*, if the `-f` flag is specified, or from standard input.

Syntax

```
ldapdelete [ -c ] [ -n ] [ -v ] [ -d debuglevel ] [ -D binddn ] [ -f file ] [ -w passwd ] \
[ -h ldaphost ] [ -p ldapport ] [ dn... ]
```

The parameters for this command are:

[`-c`]	Specifies continuous operation mode. Errors are reported, but `ldapdelete` continues with deletions. The default is to exit after reporting an error.
[`-n`]	Shows what would be done, but doesn't actually delete entries. Useful in conjunction with `-v` and `-d` for debugging.
[`-v`]	Uses verbose mode, with diagnostics written to standard output.

[`-d` *debuglevel*]	Sets the LDAP debugging level. Useful levels of debugging for `ldapmodify` and `ldapadd` are: • 1 - Trace • 2 - Packets • 4 - Arguments • 32 - Filters • 128 - Access control To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a debug level of 33. See the section entitled “`slapdcmd`” for a complete list of debugging and trace levels.
[`-D` *binddn*]	Uses the distinguished name *binddn* to bind to the directory.
[`-f` *file*]	Reads the entry deletion information from *file* instead of from standard input.
[`-w` *passwd*]	Uses *passwd* as the password for authentication to the directory.
[`-h` *ldaphost*]	Specifies an alternate host on which the `slapd` server is running.
[`-p` *ldapport*]	Specifies an alternate TCP port where the `slapd` server is listening.
[*dn*...]	Specifies one or several distinguished names of entries to delete.

Example

To delete the entry named with the common name (cn) `Delete Me` directly below the `XYZ Corporation` organizational entry, use following command:

```
% ldapdelete -D "cn=Administrator, o=XYZ, c=US" -w password \
"cn=Delete Me, o=XYZ, c=US"
```

ldapmodify

The `ldapmodify` command opens a connection to an LDAP server, binds, and modifies or adds entries. The entry information is read from standard input or from a file, specified using the `-f` option. The `ldapadd` command is a variation of the `ldapmodify` command. When invoked as `ldapadd`, the `-a` (add new entry) flag is automatically turned on. Both `ldapadd` and `ldapmodify` reject duplicate attribute-name/value pairs for the same entry.

Syntax

```
ldapmodify [ -abcrnvF ] [ -d debuglevel ] [ -D binddn ] [ -w passwd ] [ -h ldaphost ]\
[ -p ldapport ] [ -f file ]
```

```
ldapadd [ -bcnvF ] [ -d debuglevel ] [ -D binddn ] [ -w passwd ] [ -h ldaphost ]\
[ -p ldapport ] [ -f file ]
```

The parameters for these commands are:

[`-a`]	Adds new entries. The default for `ldapmodify` is to modify existing entries. If invoked as `ldapadd`, this flag is always set.
[`-b`]	Assumes that any value that starts with a forward slash (`/`) is the pathname of a file containing the actual attribute value. This is useful for attribute values in binary format.
[`-c`]	Runs in continuous operation mode. Errors are reported, but `ldapmodify` continues with modifications. The default is to exit after reporting an error.
[`-r`]	Replaces existing value with the specified value. This is the default for `ldapmodify`. When `ldapadd` is called, or if the `-a` option is specified, the `-r` option is ignored.
[`-n`]	Previews modifications, but make no changes to entries. Useful in conjunction with `-v` and `-d` for debugging.
[`-v`]	Uses verbose mode, with diagnostics written to standard output.
[`-F`]	Forces application of all changes regardless of the content of input lines that begin with `replica:`. By default, `replica:` lines are compared against the LDAP server host and port in use to decide whether a `replog` record should be applied.
[`-d` *debuglevel*]	Sets the LDAP debugging level. Useful levels of debugging for `ldapmodify` and `ldapadd` are: 1 - Trace 2 - Packets 4 - Arguments 32 - Filters 128 - Access control To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a debug level of 33. See the following section “`slapdcmd`” for a complete list of debugging and trace levels.
[`-D` *binddn*]	Uses the distinguished name *binddn* to bind to the directory.

[`-f` *file*]	Reads the entry modification information from *file* instead of from standard input.
[`-h` *ldaphost*]	Specifies an alternate host on which the `slapd` server is running.
[`-p` *ldapport*]	Specifies an alternate TCP port where the `slapd` server is listening.
[`-w` *passwd*]	Uses *passwd* as the password for authentication to the directory.

Input Format

The format of the input to `ldapadd` and `ldapmodify` is defined in "`slapdrepl`." The following exceptions to the `slapd.replog` file format are allowed:

- If the first line of a record consists of a decimal number (`entry id`), it is ignored.
- Lines that begin with `replica:` are matched against the LDAP server host and port in use to decide whether a particular `replog` record should be applied. The `-F` flag can be used to force `ldapmodify` to apply all of the `replog` changes, regardless of the presence or absence of any `replica:` lines. Any other lines that precede the `dn:` line are ignored.
- If no `changetype:` line is present, the default is `add` if the `-a` flag is set (or if the program was invoked as `ldapadd`) and `modify` otherwise.
- If the `changetype:` is `modify` and no `add:`, `replace:`, or `delete:` lines appear, the default is `replace:` if the `-r` flag is set and `add:` otherwise.

These exceptions to the `slapd.replog` format allow LDIF entries to be used as input to `ldapmodify` or `ldapadd`. For details on the LDIF format and the `ldif` command, see "`ldif2ldbm`." Information about configuring `slapd.replog` is available in Chapter 4, "Sun Directory Services Configuration," in the section entitled "Directory Service Log Files."

Examples

1. The file `/tmp/entrymods` contains the following modification instructions:

```
dn: cn=Modify Me, o=XYZ, c=US
changetype: modify
replace: mail
mail: modme@atlanta.xyz.com
-
add: title
title: System Manager
-
add: jpegPhoto
jpegPhoto: /tmp/modme.jpeg
-
delete: description
-
```

The command:

```
% ldapmodify -b -r -f /tmp/entrymods
```

modifies the `Modify Me` entry as follows:

- The current value of the mail attribute is replaced with the value `modme@atlanta.xyz.com`.
- A title attribute with the value `System Manager` is added.
- A `jpegPhoto` attribute is added, using the contents of the file `/tmp/modme.jpeg` as the attribute value.
- The description attribute is removed.

2. The file `/tmp/newentry` contains the following information for creating a new entry:

```
dn: cn=Ann Jones, o=XYZ, c=US
objectClass: person
cn: Ann Jones
cn: Annie Jones
sn: Jones
title: Director of Research and Development
mail: ajones@londonrd.xyz.us.com
uid: ajones
```

The following command adds a new entry for `Ann Jones`, using the information in the file:

```
% ldapadd -f /tmp/newentry
```

3. The file `/tmp/badentry` contains the following information about an entry to be deleted:

```
dn: cn=Ann Jones, o=XYZ, c=US
changetype: delete
```

The following command removes the entry for `Ann Jones`:

```
% ldapmodify -f /tmp/badentry
```

`ldapmodrdn`

The `ldapmodrdn` command opens a connection to an LDAP server, binds, and modifies the RDN of entries. The entry information is read from standard input, from *file* through the use of the `-f` option, or from the command-line pair *dn* and *rdn*.

Syntax

```
ldapmodrdn [ -cnrv ] [ -d debuglevel ] [ -D binddn ] [ -f file ] [ -w passwd ] \
[ -h ldaphost ] [ -p ldapport ] [ dn rdn ]
```

The parameters for this command are:

[`-c`]	Run in continuous operation mode. Errors are reported, but `ldapmodrdn` continues with modifications. The default is to exit after reporting an error.
[`-n`]	Shows what would be done, but doesn't actually change entries. Useful in conjunction with `-v` for debugging.
[`-r`]	Removes old RDN values from the entry. By default, old values are kept.

[-v]	Uses verbose mode, with diagnostics written to standard output.
[-d *debuglevel*]	Sets the LDAP debugging level. Useful values of *debuglevel* for `ldapmodrdn` are: • 1 - Trace • 2 - Packets • 4 - Arguments • 32 - Filters • 128 - Access control To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a *debuglevel* of 33. See the section entitled "`slapdcmd`" for a complete list of debugging and trace levels.
[-D *binddn*]	Uses the distinguished name *binddn* to bind to the directory.
[-f *file*]	Reads the entry modification information from *file* instead of from standard input or the command-line.
[-w *passwd*]	Uses *passwd* as the password for authentication to the directory.
[-h *ldaphost*]	Specifies an alternate host on which the `slapd` server is running.
[-p *ldapport*]	Specifies an alternate TCP port where the `slapd` server is listening.
[*dn* *rdn*]	When you specify a *dn* *rdn* pair, *rdn* is a Relative Distinguished Name that replaces the Distinguished Name of the entry specified by *dn*.

Input Format

If the command-line arguments *dn* and *rdn* are given, *rdn* replaces the RDN of the entry specified by the DN, *dn*. Otherwise, the contents of *file* (or standard input if no `-f` flag is specified) must consist of one or more pair of lines:

Distinguished Name (DN)
Relative Distinguished Name (RDN)

Use one or more blank lines to separate each DN/RDN pair.

Example

The file `/tmp/entrymods` contains:

```
cn=Modify Me, o=XYZ, c=US
cn=The New Me
```

The command:

```
% ldapmodrdn -r -f /tmp/entrymods
```

changes the RDN of the “Modify Me” entry from “Modify Me” to “The New Me” and the old `cn`, “Modify Me,” is removed.

ldapsearch

The `ldapsearch` command opens a connection to an LDAP server, binds, and performs a search using the *filter* filter. If `ldapsearch` finds one or more entries, the attributes specified by *attrs* are retrieved and the entries and values are printed to standard output. If no attributes are listed, all attributes are returned.

Syntax

```
ldapsearch [ -nuvtABLR ] [ -d debuglevel ] [ -F sep ] [ -f file ] [ -D binddn ] \
[ -w passwd ] [ -h ldaphost ] [ -p ldapport ] [ -b searchbase ] [ -s scope ] [ -S attr ] \
[ -a deref ] [ -l timelimit ] [ -z sizelimit ] filter [ attrs... ]
```

The parameters for this command are:

[`-n`]	Shows what would be done, but doesn’t actually perform the search. Useful in conjunction with `-v` and `-d` for debugging.
[`-u`]	Includes the user-friendly form of the Distinguished Name (DN) in the output.
[`-v`]	Runs in verbose mode, with diagnostics written to standard output.
[`-t`]	Writes retrieved values to a set of temporary files. This is useful for dealing with non-ASCII values such as `jpeg` photo or audio.
[`-A`]	Retrieves attributes only (no values). This is useful when you just want to see whether an attribute is present in an entry and are not interested in the specific value.
[`-B`]	Does not suppress display of non-ASCII values. This is useful when dealing with values that appear in alternate character sets such as ISO-8859.1. This option is automatically set by the `-L` option.
[`-L`]	Displays search results in LDIF format. This option also turns on the `-B` option, and causes the `-F` option to be ignored.

[`-R`]	Does not automatically follow referrals returned while searching.
[`-d` *debuglevel*]	Sets the LDAP debugging level. Useful levels of debugging for `ldapmodify` and `ldapadd` are: • 1 - Trace • 2 - Packets • 4 - Arguments • 32 - Filters • 128 - Access control To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a debuglevel of 33. See the section entitled "`slapdcmd`" for a complete list of debugging and trace levels.
[`-F` *sep*]	Uses *sep* as the field separator between attribute names and values. The default separator is an equal sign (=). If the `-L` flag has been specified, this option is ignored.
[`-f` *file*]	Reads a series of lines from *file*, performing one LDAP search for each line. In this case, the filter given on the command line is treated as a pattern where the first occurrence of % is replaced with a line from *file*. If *file* is a single character, then the lines are read from standard input.
[`-D` *binddn*]	Uses the distinguished name *binddn* to bind to the directory.
[`-w` *passwd*]	Uses *passwd* as the password for authentication to the directory.
[`-h` *ldaphost*]	Specifies an alternate host on which the `slapd` server is running.
[`-p` *ldapport*]	Specifies an alternate TCP port where the `slapd` server is listening.
[`-b` *searchbase*]	Uses *searchbase* as the starting point for the search instead of the default.
[`-s` *scope*]	Specifies the scope of the search. The possible values for *scope* are `base`, `one`, or `sub` to specify respectively a base object, one-level, or subtree search. The default is `sub`.
[`-S` *attribute*]	Sorts the entries returned base on *attribute*. The default behavior is not to sort entries returned. If attribute is a zero-length string (""), the entries are sorted by the components of their Distinguished Name. Note that `ldapsearch` normally prints out entries as it receives them. If you specify the `-S` option, all entries are retrieved, then sorted, then printed.
[`-a` *deref*]	Specifies how aliases dereferencing is done. The possible values for *deref* are `never`, `always`, `search`, or `find` to specify respectively that aliases are never dereferenced, always dereferenced, dereferenced when searching, or dereferenced only when finding the base object for the search. The default is to never dereference aliases.
[`-l` *timelimit*]	Waits, at most, *timelimit* seconds for a search to complete.

[-z *sizelimit*]	Retrieves, at most, *sizelimit* amount for a search to complete.
filter	A filter retrieves sys.log entries and interacts with selected subsets, depending on which filter is selected.
attrs	The set of attributes that are part of the log entries shown

Examples

1. The following command:

```
% ldapsearch "cn=mark smith" cn telephoneNumber
```

performs a subtree search (using the default search base) for entries with a commonName of mark smith. The commonName and telephoneNumber values are retrieved and printed to standard output. The output might look like this:

```
cn=Mark D Smith, ou=Sales, ou=Atlanta, ou=People, o=XYZ, c=US
cn=Mark Smith
cn=Mark David Smith
cn=Mark D Smith 1
cn=Mark D Smith
telephoneNumber=+1 123 456-7890

cn=Mark C Smith, ou=Distribution, ou=Atlanta, ou=People, o=XYZ, c=US
cn=Mark Smith
cn=Mark C Smith 1
cn=Mark C Smith
telephoneNumber=+1 123 456-9999
```

2. The command:

```
% ldapsearch -u -t "uid=mcs" jpegPhoto audio
```

performs a subtree search using the default search base for entries with user id of `mcs`. The user-friendly form of the entry's DN is output after the line that contains the DN itself, and the `jpegPhoto` and audio values are retrieved and written to temporary files. The output might look like this if one entry with one value for each of the requested attributes is found:

```
cn=Mark C Smith, ou=Distribution, ou=Atlanta, ou=People, o=XYZ, c=US
Mark C Smith, Distribution, Atlanta, People, XYZ, US
audio=/tmp/ldapsearch-audio-a19924
jpegPhoto=/tmp/ldapsearch-jpegPhoto-a19924
```

3. The command:

```
# ldapsearch -L -s one -b "c=US" "o=XY*" o description
```

performs a one-level search at the `c=US` level for all organizations whose `organizationName` begins with XY. Search results are displayed in the LDIF format. The `organizationName` and description attribute values are retrieved and printed to standard output, resulting in output similar to the following:

```
dn: o=XYZ, c=US
o: XYZ
description: XYZ Corporation

dn: o="XY Trading Company", c=US
o: XY Trading Company
description: Import and export specialists

dn: o=XYInternational, c=US
o: XYInternational
o: XYI
o: XY International
```

Output Format

If one or more entries are found, each entry is written to standard output in the form:

```
Distinguished Name (DN)
User Friendly Name (if the -u option is used)
attributename=value
attributename=value
attributename=value
...
```

Multiple entries are separated with a single blank line. If the `-F` option is used to specify a different separator character, this character is used instead of the '=' character. If the `-t` option is used, the name of a temporary file is returned in place of the actual value. If the `-A` option is given, only the "attributename" is returned and not the attribute value.

`ldbmcat`

The `ldbmcat` command is used to convert a `slapd` LDBM database to the LDAP Directory Interchange Format (LDIF) as defined in "`ldif2ldbm`." It opens the *id2entryfile* file for the database to be converted and writes the corresponding LDIF output to standard output.

Syntax

```
ldbmcat [ -n ] id2entryfile
```

The parameter for this command is:

`-n`	This option specifies that `ldbmcat` should not print entry IDs when it dumps the database. The printing of entry IDs is essential if you are going to use the LDIF format produced as input to `ldif2index`, for example, to create a new index file for use with an existing database.

Examples

To make a text backup of your LDBM database and put it in a file called `ldif.backup`, type the following command:

```
# ldbmcat -n id2entry.dbb > ldif.backup
```

To create a new index for the mail attribute, type these commands:

```
# ldbmcat id2entry.dbb > ldif.newind
# ldif2index -i ldif.newind -f slapd.config mail
```

Note – To ensure consistency of the database, the `slapd` daemon should not be running (or must at least not be in read-write mode) when you do this.

`ldif`

The `ldif` command converts arbitrary data to the LDAP Directory Interchange Format (LDIF). `ldif` reads data from standard input, converts it, and writes the corresponding LDIF output to standard output. The output is suitable for use as a line in an LDIF file.

By default, `ldif` considers its input a sequence of values, one value on each line, to be converted to values of the specified attribute. With the `-b` *flag*, `ldif` considers its input as a single raw binary value to be converted. This is useful when converting binary data such as a photo or audio attribute.

Syntax

```
ldif [ -b ] attrname
```

The parameter for this command is:

`-b`	This option specifies that `ldif` should interpret its input as a single binary value for conversion. If this option is not specified, it interprets it as a sequence of lines, with each line containing a single value.

`ldif2ldbm`

This section describes the following conversion utilities used to convert LDIF to LDBM database format:

- `ldif2ldbm`
- `ldif2index`
- `ldif2id2entry`
- `ldif2id2children`

These utilities convert a database in LDAP Directory Interchange Format (LDIF) to an LDBM database suitable for use by `slapd`. Normally, you need only use `ldif2ldbm`. It invokes the other utilities as necessary. Occasionally, it may be necessary to invoke them directly. For example, to create a new index file for an existing database, use the `ldif2index` program. To do the reverse conversion, from LDBM to LDIF, use the `ldbmcat` command, described in "`ldbmcat`."

Syntax

```
ldif2ldbm -c -i ldifinputfile [ -d debuglevel ] [ -f slapdconfigfile ]\
[ -j numberofjobs ]

ldif2index -i ldifinputfile [ -d debuglevel ] [ -f slapdconfigfile ] attributename

ldif2id2entry -i ldifinputfile [ -d debuglevel ] [ -f slapdconfigfile ]

ldif2id2children -i ldifinputfile [ -d debugevell ] [ -f slapd-configfile ]
```

The parameters for these commands are:

`-c`	This option is used to create and specifically overwrite an existing directory.
`-i` *ldifinputfile*	This option specifies the location of the LDIF input file containing the database to convert.
`-d` *debuglevel*	Set the LDAP debugging level. Useful values of *debuglevel* for these commands are: • 1 - Trace • 2 - Packets • 4 - Arguments • 32 - Filters • 128 - Access control • 2048 - Parse To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a *debuglevel* of 33.
`-f` *slapdconfigfile*	This option specifies the `slapd` configuration file. The default is `/etc/opt/SUNWconn/ldap/current/slapd.conf`.
`-j` *numberofjobs*	This option applies only to the `ldif2ldbm` utility. It specifies the number of processes that can run in parallel when doing the conversion. `ldif2ldbm` invokes several other utilities during the conversion process, most notably one invocation of `ldif2index` for each indexed attribute that appears in the LDIF input file. The `-j` option tells `ldif2ldbm` how many of these other utilities can run in parallel. Running several processes in parallel can speed up the conversion; however, it also consumes more disk, memory, and CPU resources. The default is to run only one process at a time.

Examples

To convert the file `ldif.input` into an LDBM database with indexes, as described in the `slapd` config file `/etc/opt/SUNWconn/ldap/current/slapd.conf`, type the following command:

```
% ldif2index -i ldif.input -f \
/opt/SUNWconn/ldap/current/slapd.conf
```

To do the same, but running two conversion subprocesses at a time, type this command:

```
% ldif2index -i ldif.input -f \
/opt/SUNWconn/ldap/current/slapd.conf -j 2
```

slapd

The `slapd` daemon is the directory server daemon. It listens for LDAP connections on port 389, responding to the LDAP operations it receives over these connections. `slapd` is typically invoked at boot time, usually out of `/etc/rc.local`. Upon startup, `slapd` normally forks and dissociates itself from the invoking tty. If the `-d` flag is specified, and debugging is set to a non-zero value, `slapd` does not fork and dissociates from the invoking `tty`.

The `slapd` daemon can be configured to provide replicated service for a data store, in conjunction with `slurpd`, the directory server update replication daemon. See "`slurpd`" for details.

Syntax

```
slapd [ -d debuglevel ] [ -f slapd-config- file ] [ -n number-of-connections ]\
[ -p port-number ] [ -s syslog-level ] [ -i ]
```

The parameters for this command are:

`-i`	This option tells `slapd` that it is being run from `inetd`, the Internet protocol daemon.
`-d` *debuglevel*	Turn on debugging as defined by *debuglevel*. If this option is specified, `slapd` does not fork or dissociate from the invoking terminal. Some general operation and status messages are printed for any value of *debuglevel*. *debuglevel* is taken as a bit string, with each bit corresponding to a different kind of debugging information.
`-f` *slapd-config-file*	Specifies the `slapd` configuration file. The default is `/etc/opt/SUNWconn/ldap/current/slapd.conf`.

`-n` *number-of-connections*	Specifies the maximum number of simultaneous connections the daemon can handle. The default is 256.
`-p` *port-number*	`slapd` listens on the default LDAP port (389) unless this option is used to specify a different port.
`-s` *syslog-level*	This option tells `slapd` at what level information should be logged to the `syslog` facility. You can request any combination of the following levels: • 1 - Trace • 2 - Packets • 4 - Arguments • 8 - Connections • 16 - BER • 32 - Filters • 64 - Configuration • 128 - Access control • 256 - Statistics (summary level) • 512 - Statistics (detailed level) • 1024 - Shell • 2048 - Parse • 65535 - All information To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a *syslog-level* of 33.

`slapdcmd`

The `slapdcmd` command sends orders to the `slapd` daemon to set the trace level, put the database into, and out of, read only mode (for backup), and get SNMP statistics about the `slapd` daemon.

Syntax

```
slapdcmd [ -t tracelevel ] [ -f | -n ] [ -s appli | assoc | dsaops |\
dsaentries | dsaint ]
```

The parameters for this command are:

`-f`	Put all data stores into read-only mode. This is usually in preparation for making a backup.
`-n`	Put all the data stores back to read-write mode.
`-t` *tracelevel*	Sets the level of information to be logged into log files. You can request any combination of the following levels: • 1 - Trace • 2 - Packets • 4 - Arguments • 8 - Connections • 16 - BER • 32 - Filters • 64 - Configuration • 128 - Access control • 256 - Statistics (summary level) • 512 - Statistics (detailed level) • 1024 - Shell • 2048 - Parse • 65535 - All information To request more than one category of debugging information, add the masks. For example, to request trace and filter information, specify a *tracelevel* of 33.

`-s appli | assoc | dsaops \`
`| dsaentries| dsaint`
	Displays the following SNMP statistics:
	• `appli` displays the application table
	• `assoc` displays all the current associations
	• `dsaops` displays the current count of LDAP operations
	• `dsaentries` displays the current count of entries and hits in the databases
	• `dsaint` displays the interactions of this `slapd` with other ldap servers during replication

slapdrepl

The `slapdrepl` command creates a replication log file for the replication daemon `slurpd` to use when creating a new replica. It extracts entry information from the data base directory (*databasedir*) and creates appropriate replica entries. All parameters are optional. If you do not supply any parameters, `slapdrepl` generates replica entries for all databases and for all replica (slave) servers.

Syntax

```
slapdrepl [ -i databasedir ] [ -v ] [ -d ] [ -c slapdconf ]\
[ -p previousconf ] [ -o replogfile ] [ -w ] [ -h replserver [ , replserver ]...\
[ , replserver ] ]
```

The parameters of this command are:

`-d`	Full debug mode. Debugging information is written to standard output.
`-v`	Use verbose mode, with diagnostics written to standard output.
`-i` *databasedir*	Create a replication log for the data store held in *databasedir*.
`-c` *slapdconf*	Specifies the `slapd` configuration file to use. The default is `/etc/opt/SUNWconn/ldap/current/slapd.conf`.
`-p` *previousconf*	The previous configuration file. If you specify a previous configuration file, replica information is generated for replicas that are new, meaning the previously saved configuration.
`-o` *replogfile*	The name of the replication log file to create. The default is defined in the `slapd` configuration.
`-w`	Overwrite existing information in the replication log file.
`-h` *replserver*	The name of the replica server for which you want to generate a replication log file. You can specify any number of replica servers.

slurpd

The `slurpd` daemon is used to propagate changes from one `slapd` database to another. If `slapd` is configured to produce a replication log, `slurpd` reads that replication log and sends the changes to the replica `slapd` instances using the LDAP protocol.

Upon startup, `slurpd` reads the replication log (given either by the `replogfile` directive in the `slapd` configuration file, or by the `-r` option). If the replication log file does not exist or is empty, `slurpd` goes to sleep. It periodically wakes up and checks to see if any changes need to be made.

When changes need to be made to replica `slapd` instances, `slurpd` locks the replication log, makes a private copy, releases the lock, and forks one copy of itself for each replica `slapd` to be updated. Each child process binds to the slave `slapd` with the DN given by the *binddn* option to the replica directive in the `slapd config` file, and sends the changes. See “`slapdrepl`” for details on the directory server daemon.

Syntax

```
slurpd [ -d debuglevel ] [ -f slapd-config- file ] [ -r slapd-replog-file ]\
[ -t temp-dir ] [ -o ]
```

The parameters for this command are:

`-o`	Run once and then exit. By default, `slurpd` processes the `replog` file, then watches for more replication entries to be appended. If the `-o` option is specified, `slurpd` processes a replication log and exits.
`-d` *debuglevel*	Turn on debugging as defined by *debuglevel*. If this option is specified, `slurpd` does not fork or dissociate from the invoking terminal. Some general operation and status messages are printed for any value of *debuglevel*. *debuglevel* is taken as a bit string, with each bit corresponding to a different kind of debugging information. See “Structure of the `imta.cnf` File” for details.
`-f` *slapd-config- file*	Specifies the `slapd` configuration file. The default is `/etc/opt/SUNWconn/ldap/current/slapd.conf`.
`-r` *slapd-replog-file*	Specifies the name of the `slapd` replication log file. By default, the name of the replication log file is read from the `slapd` configuration file. The `-r` option allows you to override this. In conjunction with the `-o` option, you can process a replication log file in a “one-shot” mode. For example, if `slurpd` has encountered errors in processing a replication log, you can run it in one-shot mode and give the rejection file name as the argument to the `-r` option, after you have resolved the problem that caused the replication failure.
`-t` *temp-dir*	`slurpd` copies the replication log to a working directory before processing it. This option allows you to specify the location of these temporary files. The default is `/usr/tmp`.

snmpslapd

The Sun Directory Server SNMP agent, `snmpslapd`, monitors the following processes:

Directory server daemon	`slapd`
Replication server daemon	`slurpd`

The Sun Directory Server SNMP agent collects statistics from these processes, for example, for the number of search requests received.

The kind of statistics an SNMP agent can collect is determined by the Management Information Bases and information objects that it supports. The `snmpslapd` agent supports the Management Information Bases (MIBs) that are part of the Messaging And Directory MANagement (MADMAN) standard. It also supports an extension to this standard.

The `snmpslapd` agent supports the Network Services Monitoring MIB (RFC 1565) that applies to all messaging and directory applications. The Network Services Monitoring MIB is composed of two sets of related objects that are organized in two tables:

- An application table (`applTable`)
- An association table (`assocTable`)

The `snmpslapd` agent returns the values of the objects (also called MIB *variables*) defined in these MIBs upon request from a management station. As an extension to the MADMAN standard, the `snmpslapd` agent also provides support for unsolicited event reports, called SNMP *traps*. When the agent detects an abnormal condition or an operating status change, it sends out a trap.

The possible values for the operating status of LDAP are: `up`, `down`, `congested`, `restarting`. Traps caused by changes in operating status contain the application name (`applName`), in this case `slapd` or `slurpd`, and the application's operating status (`applOperStatus`).

Syntax

```
snmpslapd [ -h ] [ -c config-file ] [ -i poll-interval ] [ -l log-file ]\
[ -p port ] [ -s log-size ] [ -T trace-level ]
```

The parameters for this command are:

`-h`	Use this option to list all available options.
`-c` *config-file*	This option allows you to specify the name of the configuration file you want to use instead of the default. You must use the full pathname. The default value for *config-file* is `/var/opt/SUNWconn/ldap/current/snmpslapd.conf`.
`-i` *poll-interval*	This option allows you to specify the frequency at which the SNMP agent daemon, `snmpslapd`, checks that the processes it monitors are actually running. The default value for *poll-interval* is 30 (seconds).
`-l` *log-file*	This option allows you to specify the name of the log file you want to use instead of the default. You must use the full pathname of the file. The default value for *log-file* is `/var/opt/SUNWconn/ldap/log/snmpslapd.log`.
`-p` *port*	This option allows you to specify the number of the UDP port that the agent resides on. The default value for *port* is 161.
`-s` *log-size*	This option allows you to limit the size of the log file for the agent. The default value for *log-size* is 10000 (bytes).
`-T` *trace-level*	This option allows you to specify a trace level between 0 and 4. The value 0 disables the trace facility. The values 1 to 4 included enable the trace facility and the output is sent to standard output. The default value for *trace-level* is 0.

Internet Message Transfer Agent

This section describes the Internet Message Transfer Agent (IMTA) utilities. These commands are in the `/opt/SUNWmail/imta/sbin/` directory. You need to be logged in as `root` to run the `imta start`, `imta stop`, `imta dirsync`, and `imta restart` commands. Unless mentioned otherwise, all IMTA commands should be run as `inetmail` (the postmaster account created during installation).

imta cache

The IMTA maintains a disk cache of all the messages currently stored in its queues. This cache is called the queue cache. The purpose of the queue cache is to make dequeue operations perform more by relieving master programs from having to open every message file to find out which message to dequeue and in which order.

The queue cache consists of the indexed files contained in the directory pointed at by the `IMTA_QUEUE_CACHE_DATABASE` option in the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`. Normally, the queue cache directory is called `/etc/opt/SUNWmail/imta/queue_cache`. This directory and the files it contains should be protected against world and group access and have the same uid as the directories `/var/opt/SUNWmail/imta/queue` and `/var/opt/SUNWmail/imta/log`.

imta cache -sync

The `imta cache -sync` command updates the active queue cache database by updating it to reflect all non-held message files currently present in the `/var/opt/SUNWmail/imta/queue/*` subdirectories. The `imta cache -close` command does not need to be issued in conjunction with the `imta cache -sync` command.

Note that the `imta cache -sync` utility does not remove any entry from the queue cache. The queue cache entries not corresponding to an actual queued message are silently discarded by master programs. They can also be removed using the `imta cache -rebuild` utility.

Syntax

```
imta cache -sync
```

Example

To synchronize the queue cache, for example, after renaming a message file, type the command:

```
# imta cache -sync
```

imta cache -rebuild

The `imta cache -rebuild` command creates a new, synchronized queue cache. Although the new database inherits the ownership and file protections of the queue cache, it is a good idea to check afterwards that the new queue cache directory and files have the same `uid` as the `queue` and `log` directories and that the queue cache database directory and files are protected against group and world access.

Caution – Rebuilding the queue cache database with this command should only be performed as a last resort–for example, if disk problems have corrupted your queue cache database–as it will cause loss of some information from the queue cache database. The type of information lost includes, but is not limited to, message creation dates, deferral dates, and expiration dates.

Syntax

```
imta cache -rebuild
```

Example

To build a new queue cache database, issue the commands:

```
% imta cache -rebuild
% imta cache -close
```

Next, the queue cache needs to be synchronized. To do this, enter the following:

```
% imta cache -sync
```

imta cache -close

The `imta cache -close` command forces IMTA processes to close any open I/O channels to the queue cache database. This is generally done for two reasons: to close all channels to the files in the database so that the database can be modified, and to force processes to reopen the queue cache database files, to begin using any new version of that database.

Syntax

```
imta cache -close
```

Example

After a new queue cache database is built with `imta cache -rebuild`, issue an `imta cache -close` command to force any detached processes to begin using the new database:

```
# imta cache -rebuild
# imta cache -close
```

imta cache -view

The `imta cache -view` command shows the current non-held entries in the IMTA cache database for a channel.

Syntax

```
imta cache -view channel-name
```

The option for this command is:

channel-name	Name of the channel for which to show entries

Example

To view entries in the queue cache database for the `tcp_local` channel, execute the command:

```
% imta cache -view tcp_local
```

imta chbuild

The `imta chbuild` command compiles the character set conversion tables and loads the resulting image file into shared memory. The IMTA ships with complete character set tables so you would not normally need to run this command.

Syntax

```
imta chbuild options
```

The options for this command are:

`-image_file=`*file-spec* `-noimage_file`	By default, `imta chbuild` creates as output the image file named by the `IMTA_CHARSET_DATA` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`. With the `-image_file` option, an alternate file name may be specified. When the `-noimage_file` option is specified, `imta chbuild` does not produce an output image file. This qualifier is used in conjunction with the `-option_file` option to produce as output an option file that specifies table sizes adequate to hold the tables required by the processed input files.
`-maximum` `-nomaximum`	The file `/etc/opt/SUNWmail/imta/maximum_charset.dat` is read in addition to the file named by the `IMTA_CHARSET_OPTION_FILE` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`, when `-maximum` is specified. This file specifies near-maximum table sizes but does not change any other file parameter settings. Use this option *only* if the current table sizes are inadequate. The `-noimage` and `-option_file` options should always be used in conjunction with this option—it makes no sense to output the enormous configuration that is produced by `-maximum`, but it does make sense to use `-maximum` to get past size restrictions in order to build a properly sized option file for use in building a manageable configuration with a subsequent `imta chbuild` invocation.

`-option_file=`*file-spec* `-nooption_file`	`imta chbuild` can produce an option file that contains the correct table sizes to hold the conversion tables that were just compiled (plus a little room for growth). The `-option_file` option causes this file to be output. By default, this file is the file named by the `IMTA_CHARSET_OPTION_FILE` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`. The value of the `-option_file` option may be used to specify an alternate file name. If the `-nooption_file` option is given, then no option file is output. `imta chbuild` always reads any option file (for example, the file named by the `IMTA_OPTION_FILE` option of the IMTA tailor file) that is already present; use of this option does not alter this behavior. However, use of the `-maximum` option causes `imta chbuild` to read options from `maximum_charset.dat` in addition to `IMTA_CHARSET_OPTION_FILE`. This file specifies near-maximum table sizes. Use this option *only* if the current table sizes are inadequate, and only use it to create a new option file. The `-noimage_file` option should always be specified with `-maximum`, since a maximum-size image would be enormous and inefficient.
`-remove`	Remove any existing compiled character set conversion table, for example, the file named by the `IMTA_CHARSET_DATA` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`.
`-sizes` `-nosizes`	The `-sizes` option instructs `imta chbuild` to output or suppress information on the sizes of the uncompiled conversion tables.
`-statistics` `-nostatistics`	The `-statistics` option instructs `imta chbuild` to output or suppress information on the compiled conversion tables. These numbers give a rough measurement of the efficiency of the compilation, and may indicate whether or not an additional rebuild with the `-option_file` option is needed.

Example

The standard command you use to compile character set conversion tables is:

```
% imta chbuild
```

imta cnbuild

The `imta cnbuild` command compiles the textual `configuration`, `option`, `mapping`, `conversion`, and `alias` files, and loads the resulting image file into shared memory. The resulting image is saved to a file usually named `/opt/SUNWmail/imta/lib/config_data` by the `IMTA_CONFIG_DATA` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`.

Whenever a component of the IMTA (for example, a channel program) must read a compiled configuration component, it first checks to see whether the file named by the IMTA tailor file option `IMTA_CONFIG_DATA` is loaded into shared memory; if this compiled image exists but is not loaded, the IMTA loads it into shared memory. If the IMTA finds (or not finding, is able to load itself) a compiled image in shared memory, the running program uses that image. This rule has two exceptions:

1. The first is `imta cnbuild` itself, which always reads the text files and never tries to use an image form of the configuration data.
2. The second exception is `imta test -rewrite`, which can be instructed with the `-image_file` option to use a different compiled configuration file. This facility in `imta test -rewrite` is useful for testing changes prior to compiling them.

The reason for compiling configuration information is simple: performance. The only penalty paid for compilation is the need to recompile and reload the image any time the configuration or alias files are edited. Also, be sure to restart any programs or channels that load the configuration data only once when they start up–for example, the IMTA multithreaded TCP SMTP server.

It is necessary to recompile the configuration every time changes are made to any of the following files:

- IMTA configuration file (or any files referenced by it)
- IMTA system alias file, the IMTA mapping file
- IMTA option file
- IMTA conversion file

Specifically, these are the files pointed at the IMTA tailor file options `IMTA_CONFIG_FILE`, `IMTA_ALIAS_FILE`, `IMTA_MAPPING_FILE`, `IMTA_OPTION_FILE`, and `IMTA_CONVERSION_FILE`, respectively, which usually point to the following files:

- `/etc/opt/SUNWmail/imta/imta.cnf`
- `/etc/opt/SUNWmail/imta/aliases`
- `/etc/opt/SUNWmail/imta/mappings`
- `/etc/opt/SUNWmail/imta/option.dat`
- `/etc/opt/SUNWmail/imta/conversions`

Note – Until the configuration is rebuilt, changes to any of these files are not visible to the running IMTA system.

Syntax

```
imta cnbuild [-remove] [-resize_tables] [-sizes] [-statistics]
```

The options for this command are:

`-resize_tables`	This option is used to get past size restriction in order to build a properly-sized option file so that a proportionately-sized configuration can be built with a subsequent `imta cnbuild` invocation. Use this option only if the current table sizes are inadequate, and you need to create a new option file. To be precise, use it only if you get a "No room in table" error when building the configuration or starting the IMTA.
`-remove`	Remove any existing compiled configuration; for example, remove the file named by the `IMTA_CONFIG_DATA` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`.
`-sizes`	The `-sizes` option instructs `imta cnbuild` to output information on the sizes of uncompiled IMTA tables.
`-statistics`	The `-statistics` command instructs `imta cnbuild` to output information on how much of the various tables in the compiled configuration were actually used to store data. These numbers give a rough measurement of the efficiency of the compilation, and may indicate whether or not an additional rebuild with the `-resize_tables` option is needed.

Examples

The standard command you use to regenerate a compiled configuration is:

```
% imta cnbuild
```

After compiling the configuration, restart any programs that may need to reload the new configuration. For example, the TCP SMTP server should be restarted:

```
% imta restart dispatcher
```

Execute the following two commands if you encounter the `No room in table:` error message:

```
% imta cnbuild -resize_tables
# imta restart
```

Note – By default, `imta cnbuild` is executed whenever the `imta start` or `imta restart` commands are invoked.

---

imta counters -clear

The IMTA accumulates in the form of message traffic statistics for each of its channels. These statistics are referred to as channel *counters*. The counters are kept in a shared memory cache.

The `imta counters -clear` command clears the in-memory channel counters..

Syntax

```
imta counters -clear
```

imta counters -create

The `imta counters -create` command creates an in-memory cache of channel counters.

Note – Do not execute this utility if you already have in-memory counters because `imta start` creates this section. Normally this utility should never be used unless you have manually deleted the counters using `imta counters -delete`.

Syntax

```
imta counters -create -max_channels=value
```

The option for this command is:

`-max_channels=`*value*	By default, the created cache can hold information for `CHANNEL_TABLE_SIZE` channels. `CHANNEL_TABLE_SIZE` is the value specified by the IMTA Option file option of the same name. Use the `-max_channels=`*value* option to select a different size

imta counters -delete

Use the `imta counters -delete` command to delete the in-memory cache of channel counters.

Syntax

```
imta counters -delete
```

imta counters -show

The contents of the in-memory cache of channel counters may be displayed with the `imta counters -show` command.

Syntax

```
imta counters -show -headers (-noheaders) -output=file-spec
```

The options for this command are:

`-header` `-noheaders`	Controls whether or not a header line describing each column in the table of counters is output. The `-headers` option is the default.

`-output=`*file-spec*	Directs the output to the specified file. By default, the output appears on your display.

Example

To display the counters for all channels, execute the following:

```
% imta counters -show
```

Output for showing the counters would look similar to the following:

```
Channel                      Messages   Recipients     Blocks
1
 Received                         656          658       1263
 Stored                            15           15         69
 Delivered                        642          643       1194
 Submitted                        998         1004       1608
reprocess
 Received                         273          279       3344
 Store                              0            0       2444
 Delivered                        273          279        900
 Submitted                          0            0          0
sims-ms
 Received                       32893        37158     242709
 Stored                             0            0     188405
 Delivered                      32893        37158      54304
 Submitted                        543          546       4250
sims-ms_append
 Received                          55           55        149
 Stored                             0            0         69
 Delivered                         55           55         80
 Submitted                         55           55        149
tcp_default_router
  Received                        604          874      11965
  Stored                            0            0         -4
  Delivered                       604          872      11965
  Submitted                         6            8         26
```

```
Channel                     Messages    Recipients     Blocks

  Total Out Assocs               589
  Rejected Out Assoc               9

tcp_local
  Received                       660           712       2122
  Stored                          -1            -1         -7
  Delivered                      661           713       2129
  Submitted                    33539         38123     255519

  Total In Assocs              33745
  Total Out Assocs               622
  Rejected Out Assocs              7
```

Note – Showing counters in this manner does *not* provide the administrator with exact numbers, but is intended to indicate trends in usage. For precise figures, use `imta cache -view`.

imta crdb

The `imta crdb` command creates and updates IMTA database files. `imta crdb` converts a plain text file into IMTA database records; from them, it either creates a new database or adds the records to an existing database.

In general, each line of the input file must consist of a left side and a right side. The two sides are separated by one or more spaces or tabs. The left side is limited to 32 characters in a short database (the default variety) and 80 characters in a long database. The right side is limited to 80 characters in a short database and 256 in a long database. Spaces and tabs may not appear in the left side.

Syntax

```
imta crdb options input-file-spec output-database-spec
```

The *options* for this command are:

`-append` `-noappend`	When the default, `-noappend`, option is in effect, a new database is created, overwriting any old database of that name. Use the `-append` option to instruct the IMTA to instead add the new records to an existing database. The `-noappend` option is the default.
`-count` `-nocount`	Controls whether or not a count is output after each group of 100 input lines are processed. The `-count` option is the default.
`-duplicates` `-noduplicates`	Controls whether or not duplicate records are allowed in the output files. Currently, duplicate records are of use only in the domain database (rewrite rules database) and databases associated with the directory channel. The `-noduplicates` option is the default.
`-long_records` `-nolong_records`	Controls the size of the output records. By default, left sides are limited to 32 characters and right sides are limited to 80 characters. If `-long_records` is specified, the limits are changed to 80 and 256, respectively. The `-nolong_record`s option is the default.
`-quoted` `-noquoted`	Controls the handling of quotes. Normally `imta crdb` pays no attention to double quotes. If `-quoted` is specified, `imta crdb` matches up double quotes in the process of determining the break between the left and right hand sides of each input line. Spaces and tabs are then allowed in the left side if they are within a matching pair of quotes. This is useful for certain kinds of databases, where spaces may form a part of database keys. The quotes are not removed unless the `-remove` option is also specified. The `-noquoted` option is the default.
`-remove` `-noremove`	Controls the removal of quotes. If `imta crdb` is instructed to pay attention to quotes, the quotes are normally retained. If `-remove` is specified, `imta crdb` removes the outermost set of quotes from the left hand side of each input line. Spaces and tabs are then allowed in the left side if they are within a matching pair of quotes. This is useful for certain kinds of databases, where spaces may form a part of database keys. `-remove` is ignored if `-quoted` is not in effect. The `-noremove` option is the default.

-statistics -nostatistics	Controls whether or not some simple statistics are output by imta crdb, including the number of entries (lines) converted, the number of exceptions (usually duplicate records) detected, and the number of entries that could not be converted because they were too long to fit in the output database. -nostatistics suppresses output of this information. The -statistics option is the default.
-strip_colons -nostrip_colons	Instructs imta crdb to strip a trailing colon from the right end of the left hand side of each line it reads from the input file. This is useful for turning alias file entries into an alias database. The -nostrip_colons is the default.

The input and output parameters for this command are:

input-file-spec	A text file containing the entries to be placed into the database. Each line of the text file must correspond to a single entry.
output-database-spec	The initial name string of the files to which to write the database. The database consists of three files named *output-database-spec*.idx, *output-database-spec*.dat, and *output-database-spec*.lock, respectively.

Example

The following commands create an alias database with “long” record entries. The creation is performed in a two-step process using a temporary database to minimize any window of time, such as during database generation, when the database would be locked and inaccessible to the IMTA.

```
% imta crdb -long_records /var/opt/SUNWmail/imta/aliases.txt \
/var/opt/SUNWmail/imta/tmpdb
%imta renamedb /var/opt/SUNWmail/imta/tmpdb IMTA_ALIAS_DATABASE
```

imta dirsync

The imta dirsync utility recreates or updates the IMTA directory cache.

Note – You must be logged in as root to use this utility.

Syntax

```
imta dirsync [ -d domain_name1, domain_name2,... ] [ -v ] \
[ -l localhost_name1, localhost_name2,... ] [ -F ] [ -L ] [ -b base-dn ]\
[ -h ldaphost1, ldaphost2, ... ] [ -p ldapport ] [ -t ]
```

The options for this command are:

`-v`	Runs this command in verbose mode. A trace file is created in the log directory.
`-F`	Performs a *full* synchronization. By default, the `imta dirsync` command performs an incremental synchronization of the directory cache, which means that only entries that have been added or modified in the directory since the last synchronization are updated. The `-F` option causes the directory cache to be completely regenerated, thus creating a faithful image of the directory. *The MTA is restarted after a full synchronization.*
`-L`	This option causes the synchronization program to cache only local entries, that is addressees whose mail server is this MTA. A directory cache is built according to the MTA role. By default, the cache holds information about all addressees in the domains specified in the admin server configuration or with the `-d` option.
`-d` *domain_name1*, *domain_name2*, . . .	Specifies the fully qualified domain names containing addressees the MTA has to know in order to route messages. If two or more domain names are given, they are separated by commas. If none is specified, the mail server's domain name stored by the admin server is used.
`-l` *localhost_name1*, *localhost_name2*, . . .	Specifies the fully-qualified host names of this MTA. If more than one host name is given, they are separated by commas. If none is specified, the official hostname defined in the MTA configuration is used.
`-b` *base_dn*	Normally, the domain name given by the `-d` option is sufficient to determine the base distinguished name (DN) under which all the entries searched are located. This option lets you manually specify this base domain.
`-h` *ldaphost1, ldapthost2, ...*	Specifies the directory hostname. The program connects to the first *ldaphost* it can connect to successfully.
`-p` *ldapport*	Specifies the LDAP port number.
`-t`	Execute `imta dirsync` in the test mode. Searches the directory and prints out the details on invalid entries, if there are any. No changes are made to the cache itself. For details on all entries, test also in verbose mode (run both the `-t` and -v options).

Example

To perform a full directory cache synchronization of `mail.domain.com`, which is a router MTA in the domain `domain.com`, execute the following command:

```
# imta dirsync -F -d domain.com
```

The following entries are synchronized only when performing a full synchronization:

- Deleted users
- Deleted distributions lists

imta dumpdb

The `imta dumpdb` command writes the entries in IMTA databases to a flat ASCII file. In particular, this command may be used to write the contents of an old style database to a file from which a new style database may be built using the `imta crdb` command. If you do not have write permission in the directory where you are working, the `imta dumpdb` command fails with the following warning:

```
# dumpdb can be executed by inetmail only
```

Note – Make sure you are logged in as `inetmail` (the postmaster) before performing this command.

Syntax

```
imta dumpdb input-database-spec  output-file-spec
```

The parameters for this command are:

input-database-spec	Database from which to read entries. By default, the IMTA looks for a current format database of the given name; if this does not exist, the IMTA will look for an old format database of the given name. The special keywords `IMTA_ALIAS_DATABASE`, `IMTA_REVERSE_DATABASE`, and `IMTA_GENERAL_DATABASE` are supported; the use of such a special keyword tells the IMTA to dump the database specified by the corresponding IMTA tailor file option.
output-file-spec	ASCII file to which the entries stored in the database are written. This file should be in a directory where you have write permissions.

Examples

The following commands can be used to dump the contents of an alias database to a file, and then to recreate the alias database from that file. Type:

```
% imta dumpdb IMTA_ALIAS_DATABASE alias.tmp
% imta crdb alias.tmp IMTA_ALIAS_DATABASE
```

imta process

This command displays the current IMTA processes. In the Enterprise Edition, the IMTA Service Dispatcher and the IMTA Job Controller and SMTP should be present; in the Departmental Edition, the IMTA Job Controller should be present. Additional processes may be present if messages are currently being processed, or if certain additional IMTA components are in use.

Syntax

```
imta process
```

Example

The following command shows current IMTA processes:

```
# imta process
inetmail 13573     1  0 17:00:03 ?        0:00
/opt/SUNWmail/imta//lib/job_controller inetmail 13582     1  0
17:00:03 ?
  0:01 /opt/SUNWmail/imta//lib/dispatcher inetmail 13584     1  0
17:00:04 ?
     0:01 <SMTP>
```

`imta program`

The `imta program` commands are used to manipulate the program delivery options.

These commands can be executed as `root` or `inetmail`. A change in an existing one will take effect only after the next full `dirsync` is performed.

Syntax

```
imta program -a -m method -p program [-g argument_list] [-e exec_permission]
imta program -d -m method
imta program -c -m method -p program | -g argument_list | -e exec_permission
imta program -u -m method [ -h ldaphost ]
imta program -l
```

The options for this command are:

`-a`	Add a method to the set of program delivery methods. This option cannot be used with the `-d`, `-c`, `-l`, or `-u` options.
`-c`	Change the arguments to a program that has already been entered.
`-m` *method*	Name given by the administrator to a particular method. This will be the name by which the method will be advertised to users. Method names must not contain spaces, tabs, or equal signs (=). The method name cannot be `none` or `locale`. This option is required with the `-a`, `-d`, `-c`, and `-u` options.
`-p` *program*	Actual name of the executable for a particular method. The executable should exist in the `programs` directory (`/opt/SUNWmail/imta/programs`) for the add to be successful. It can be a symbolic link to an executable in some other directory. This option is required with the `-a` option.
`-g` *argument_list*	Argument list to be used while executing the program. If this option is not specified during an add, no arguments will be used. Each argument must be separated by a space and the entire argument list must be given within double quotes. If the `%s` tag is used in the argument list, it will be substituted with the user's username for programs executed by the users and with `username+`*programlabel* for programs executed by the postmaster `inetmail`. *programlabel* is a unique string to identify that program. This option can be used with the `-a` and `-c` options.
`-e` *exec_permission*	*exec_permission* can be `user` or `postmaster`. If it is specified as user, the program is executed as the user. By default, execute permission for all programs are set to `postmaster`. Programs with *exec_permission* set to `user` can be accessed by users with UNIX accounts only. This option can be used with the `-a` and `-c` options.
`-d`	Delete a method from the list of supported program delivery methods. This option *cannot* be used with the `-a`, `-c`, `-l`, or `-u` options.

`-u`	List the users that use a particular method. This option *cannot* be used with the `-d`, `-c`, `-l`, or `-a` options.
`-h` *ldaphost*	Specify the directory hostname if different from localhost. This option can be used with the `-u` option only.
`-l`	List all supported program delivery methods. This option *cannot* be used with the `-d`, `-c`, `-a`, or `-u` options.

Examples

To add a method `procmail1` that executes the program `procmail` with the arguments `-d` *username* and executes as the user, enter the following:

```
% imta program -a -m procmail1 -p procmail -g "-d %s" -e user
```

To list all the methods defined:

```
% imta program -l
```

To list all users executing the method `procmail1`:

```
% imta program -u -m procmail1
```

imta purge

The `imta purge` command deletes older versions of IMTA log files. `imta purge` can determine which log files are older, based on the `uniqueid` strings terminating IMTA log file names.

Syntax

```
imta purge [file-pattern] -day=d-value -hour=h-value -num=n-value
```

If specified, the *file-pattern* parameter is a filename pattern that establishes which IMTA log files to purge. The default pattern, if none is specified, is `/var/opt/SUNWmail/imta/log`. The options for this command are:

`-day=`*d-value*	Purges all but the last *d-value* days worth of log files.
`-hour=`*h-value*	Purges all but the last *h-value* hours worth of log files.
`-num=`*n-value*	Purges all but the last *n-value* log files. The default is 5.

Example

To purge all but the last five versions of each type of log file in `/var/opt/SUNWmail/imta/log`, execute the command:

```
# imta purge
```

imta queue

The `imta queue` command is used to perform common maintenance tasks on the IMTA message queues. Unlike the `imta cache` utility, operations performed with `imta queue` apply not only to the queue cache database but also to the actual message queues (message files).

Syntax

```
imta queue -recover_crash | -retry_delivery channel_name
```

The options for this command are:

`-recover_crash`	`imta -recover_crash` rebuilds the IMTA queue-cache database after a crash.
`-retry_delivery` *channel_name*	`imta queue -retry_delivery` reprocesses the HELD messages in the channel specified by the *channel_name* parameter. In order to avoid mail loops, the IMTA holds messages when they have been forwarded more than 30 times. When the reason for the mail loop has been detected and corrected, the administrator can run this command to reprocess all the HELD messages.

imta renamedb

The `imta renamedb` command renames an IMTA database. Since the IMTA may optionally reference several “live” databases, that is, databases whose presence triggers their use by the IMTA, it is important, first, to ensure that the IMTA does not see such a database while it is in a mixed state, and second, to minimize any period of time during which the database is inaccessible. The `imta crdb` command locks the database it is creating to avoid having it accessed in a mixed state.

It is recommended that the IMTA databases be created or updated in a two-step process:

1. Create or update a temporary database
2. Rename the temporary database with the “live” name using the `imta renamedb` command.

The `imta renamedb` utility, which must delete any old database files and rename the new database files, locks the database during the renaming process to avoid presenting the database in a mixed state. In this way the database is never accessible while it is in a mixed state, yet any window of time during which the database is inaccessible is minimized. Renaming is generally quicker than database generation.

Syntax

```
imta renamedb old-database-spec  new-database-spec
```

The parameters for this command are:

old-database-spec	The name of the database that is being renamed.
new-database-spec	The new name of the database. This may either be an actual pathname, or one of the special names such as `IMTA_ALIAS_DATABASE`, `IMTA_REVERSE_DATABASE`, `IMTA_GENERAL_DATABASE`, or `IMTA_DOMAIN_DATABASE`, listed in the IMTA tailor file and pointing to actual pathnames.

Example

The following command renames the database `tmpdb` to be the actual IMTA alias database (usually `/var/opt/SUNWmail/imta/db/aliasesdb`). Type:

```
% imta renamedb tmpdb IMTA_ALIAS_DATABASE
```

imta restart

The `imta restart` command stops any IMTA Job Controller or IMTA Service Dispatcher jobs that are running, and restarts the IMTA Job Controller and IMTA Service Dispatcher. Detached IMTA processes should be restarted whenever the IMTA configuration is altered—these processes load information from the configuration only once and need to be restarted in order for configuration changes to become visible to them. In addition to general IMTA configuration files, such as the `imta.cnf` file, some components, such as the IMTA Service Dispatcher, have their own specific configuration files, for example, `dispatcher.cnf`, and should be restarted after changes to any of these files. The `dispatcher.cnf` file is only available in the Sun Internet Mail Server 3.5 - Enterprise Edition.

Note – You must be logged in as `root` to use this utility.

Syntax

```
imta restart [component]
```

component is an optional parameter that identifies a specific IMTA component to be restarted, for example, `job_controller` or `dispatcher`. Restarting the IMTA Service Dispatcher effectively restarts all the service components it handles. If no component name is given, all active components are restarted.

Example

To restart the IMTA jobs, type:

```
# imta restart
```

imta return

The `imta return` command returns a message to the message's originator. The returned message is in two parts. The first part explains the reason why the message is being returned. The text of the reason is contained in the file `return_bounce.txt` located in the `/etc/opt/SUNWmail/imta/locale/C/LC_MESSAGES/` directory. The second part of the returned message contains the original message.

Syntax

```
imta return message-file
```

message-file is the name of the message file to return. The name may include wildcards, but if so, the specification must be quoted.

Example

The following command causes all of the messages currently in the local channel l (lowercase letter “l”), to be returned to their respective originators. Type:

```
# imta return ‘imta/queue/1/*’
```

imta run

The `imta run` command processes the messages in the channel specified by the *channel* parameter. Output during processing is displayed at your terminal, which makes your terminal unavailable for the duration of the operation of the utility. Refer also to the `imta submit` command that, unlike `imta run`, does not monopolize your terminal.

Syntax

```
imta run channel [poll]
```

The parameters for this command are:

channel	Specifies the channel to be processed. This parameter is not optional.
`poll`	If `poll` is specified, the channel program only attempts to run if actual messages for that channel are waiting to be processed.

Example

Type the following command to process any messages in the `tcp_local` channel:

```
% imta run tcp_local poll
```

imta start

The `imta start` command starts up detached IMTA processes. If no component parameter is specified, then the IMTA Job Controller and (in the Enterprise Edition) IMTA Service Dispatcher are started. Starting the Service Dispatcher starts all services the Service Dispatcher is configured to handle, which may include SMTP server. If a component parameter is specified, then only detached processes associated with that component are started. The standard component names are:

Component	Description
`dispatcher`	Multithreaded Service Dispatcher
`job_controller`	Schedules deliveries (dequeues messages).

The services handled by the IMTA multithreaded Service Dispatcher must be started by starting the IMTA Service Dispatcher. Only services not being handled by the IMTA Service Dispatcher can be individually started via the `imta start` command. The Service Dispatcher may be configured to handle various services, for example, the multithreaded SMTP server.

Note – You must be logged in as `root` to use this utility.

Syntax

```
imta start [component-name]
```

Example

Use the following command to start the IMTA Job Controller and IMTA Service Dispatcher:

```
# imta start
```

`imta stop`

The `imta stop` command shuts down the IMTA Job Controller and the IMTA Service Dispatcher. Shutting down the IMTA Service Dispatcher shuts down all services (for example, SMTP) being handled by the Service Dispatcher.

Note – You must be logged in as `root` to use this utility.

Syntax

```
imta stop [component-name]
```

Example

Use the following command to shut down the IMTA jobs:

```
# imta stop
```

Note – You must have `root` privileges to run this command.

`imta submit`

The `imta submit` command forks a process to execute the messages in the channel specified by the *channel* parameter.

Syntax

```
imta submit [channel] [poll]
```

The parameters for this command are:

channel	Specifies the channel to be processed. The default, if this parameter is not specified, is the local channel 1.
`poll`	If `poll` is specified, the channel program only attempts to run if actual messages for that channel are waiting to be processed.

Example

Use the following command to process any messages in the `tcp_local` channel:

```
% imta submit tcp_local poll
```

imta test -rewrite

`imta test -rewrite` provides a test facility for examining the IMTA's address rewriting and channel mapping process without actually sending a message. Various qualifiers can be used to control whether `imta test -rewrite` uses the configuration text files or the compiled configuration (if present), the amount of output produced, and so on.

If a test address is specified on the command line, `imta test -rewrite` applies the IMTA address rewriting to that address, reports the results, and exits. If no test address is specified, `imta test -rewrite` enters a loop, prompting for an address, rewriting it, and prompting again for another address. `imta test -rewrite` exits when CTRL-D is entered.

When testing an email address corresponding to a restricted distribution list, `imta test -rewrite` uses as the posting address the return address of the local postmaster, which is usually `postmaster@localhost` unless specified by the IMTA option `RETURN_ADDRESS` in the IMTA Option file.

Syntax

```
imta test -rewrite options
```

The options for this command are:

`-address=`*address*	The argument to the `-address` option specifies the test address to be rewritten. If this option is omitted, then `imta test -rewrite` prompts for an address.
`-alias_file=`*filename*	Specifies an alternate file for `imta test -rewrite` to use. `imta test -rewrite` normally consults the default alias file named by the `IMTA_ALIAS_FILE` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`, during the rewriting process. This option has no effect unless `-noimage_file` is specified or no compiled configuration exists; any compiled configuration precludes reading any sort of alias file.
`-check_expansions` `-nocheck_expansions`	Controls checking of alias address expansion. Normally the IMTA considers the expansion of an alias to have been successful if any of the addresses to which the alias expands are legal. The `-check_expansions` option causes a much stricter policy to be applied: `imta test -rewrite -check_expansions` checks each expanded address in detail and reports a list of any addresses, expanded or otherwise, that fail to rewrite properly.
`-configuration_file=`*file*	Specifies an alternate file to use in place of the file named by `IMTA_CONFIG_FILE`. Normally, `imta test -rewrite` consults the default configuration file named by the `IMTA_CONFIG_FILE` option of the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`, during the rewriting process. This option has no effect unless `-noimage_file` is specified or no compiled configuration exists; any compiled configuration precludes reading any sort of configuration file.
`-database=`*database-list*	Disables references to various databases or redirects the database paths to nonstandard locations. `imta test -rewrite` normally consults the usual IMTA databases during its operation. The allowed list items are `alias`, `noalias`, `domain`, `nodomain`, `general`, `nogeneral`, `reverse`, and `noreverse`. The list items beginning with "no" disable use of the corresponding database. The remaining items require an associated value, which is taken to be the name of that database.

`-debug` `-nodebug`	Enables the production of the additional, detailed explanations of the rewriting process.This option is disabled by default.
`-delivery_receipt` `-nodeliver_receipt`	Sets the corresponding receipt request flags. These options can be useful when testing the handling of sent or received receipt requests when rewriting forwarded addresses or mailing lists.
`-destination_channel=`*channel*	Controls to which destination or target channel `imta test -rewrite` rewrites addresses. Some address rewriting is destination channel specific; `imta test -rewrite` normally pretends that its channel destination is the local channel l.
`-grey=`*setting*	Controls the setting of the Grey Book flag. By default, this flag has a value of 0.
`-image_file=`*filename* `-noimage_file`	Instructs `imta test -rewrite` to unconditionally ignore any previously compiled configuration and to read configuration information from the various text files instead. When the `-image_file` option is specified without an optional file name, `imta test -rewrite` loads the compiled configuration from the file named by the `IMTA_CONFIG_DATA` option into the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`, which is usually `/etc/opt/SUNWmail/imta/imta.cnf`. If, instead, a file name is specified, then `imta test -rewrite` loads the compiled configuration from the specified file.
`-input=`*input-file*	Specifies a source for input to `imta test -rewrite`. By default, `imta test -rewrite` takes input from `stdin`.
`-local_alias=`*value* `-nolocal_alias`	Controls the setting of an alias for the local host. The IMTA supports multiple “identities” for the local host; the local host may have a different identity on each channel. This option may be used to set the local host alias to the specified value; appearances of the local host in rewritten addresses are replaced by this value.
`-mapping_file=`*file* `-nomapping_file`	Instructs `imta test -rewrite` to use the specified mapping file rather than the default mapping file named by the `IMTA_MAPPING_FILE` option in the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`, which is usually the file named `/etc/opt/SUNWmail/imta/mappings`. This option has no effect unless `-noimage_file` was specified or no compiled configuration exists; any compiled configuration precludes reading the mappings file. Use of the `-nomapping_file` option will prevent the `IMTA_MAPPING_FILE` file from being read in when there is no compiled configuration.

`-option_file=`*filename* `-nooption_file`	Instructs `imta test -rewrite` to use the specified option file rather than the default option file named by the `IMTA_OPTION_FILE` option in the IMTA tailor file, `/etc/opt/SUNWmail/imta/imta_tailor`, which is usually the file `/etc/opt/SUNWmail/imta/options.dat`. This option has no effect unless `-noimage_file` is specified or no compiled configuration exists; any compiled configuration precludes reading any sort of option file. Use of the `-nooption_file` option prevents the `IMTA_OPTION_FILE` file from being read in when there is no compiled configuration.
`-output=`*output_file*	Directs the output of `imta test -rewrite`. By default, `imta test -rewrite` writes output to `stdout`.
`-read_receipt` `-noread_receipt`	Sets the corresponding receipt request flags. This option can be useful when testing the handling of receipt requests at the time of rewriting forwarded addresses or mailing lists.
`-restricted=`*setting*	Controls the setting of the restricted flag. By default, this flag has value 0. When set to 1, `-restricted=1`, the restricted flag is set on and addresses are rewritten using the restricted mailbox encoding format recommended by RFC 1137. This flag is used to force rewriting of address mailbox names in accordance with the RFC 1137 specifications.
`-source_channel=`*channel*	Controls for which source channel rewrites addresses. Some address rewriting is source channel-specific; `imta test -rewrite` normally assumes that the channel source for which it is rewriting is the local channel l.

Example

This example shows typical output generated by `imta test -rewrite`. The most important piece of information generated by `imta test -rewrite` is displayed on the last few lines of the output, which shows the channel to which `imta test -rewrite` would submit a message with the specified test address and the form in which the test address would be rewritten for that channel. This output is invaluable when debugging configuration problems.

```
# imta test -rewrite
Address: joe.blue
channel                = l
  channel description    =
  channel flags #1       = BIDIRECTIONAL MULTIPLE IMMNONURGENT
NOSERVICEALL
  channel flags #2       = NOSMTP POSTHEADBODY HEADERINC NOEXPROUTE
  channel flags #3       = LOGGING NOGREY NORESTRICTED
  channel flags #4       = EIGHTNEGOTIATE NOHEADERTRIM NOHEADERREAD RULES
  channel flags #5       =
  channel flags #6       = LOCALUSER NOX_ENV_TO RECEIPTHEADER
  channel flags #7       = ALLOWSWITCHCHANNEL NOREMOTEHOST DATEFOUR
DAYOFWEEK
  channel flags #8       = NODEFRAGMENT EXQUOTA REVERSE
NOCONVERT_OCTET_STREAM
  channel flags #9       = NOTHURMAN INTERPRETENCODING
  text/plain charset def = (7) US-ASCII 5 (8) ISO-8859-1 51
  channel envelope address type = SOURCEROUTE
  channel header address type = SOURCEROUTE
  channel official host  = mailserver.eng.alpha.com
  channel local alias    =
  channel queue name     =
  channel after param    =
  channel daemon name    =
  channel user name      =
  notices                =
  channel group ids      =
  header To: address     = joe.blue@mailserver.eng.alpha.com
  header From: address   = joe.blue@mailserver.eng.alpha.com
  envelope To: address   = joe.blue@mailserver.eng.alpha.com  (route
(mailserver.eng.alpha.com,mailserver.eng.alpha.com))
  envelope From: address = joe.blue@mailserver.eng.alpha.com
  name                   =
  mbox                   = joe.blue
Extracted address action list: joe.blue@mailserver.eng.alpha.com
Extracted 733 address action list: joe.blue@mailserver.eng.alpha.com
Expanded address:
  joe.blue@mailserver.eng.alpha.com
Submitted address list:
  sims-ms
    joe.blue@sims-ms-daemon (sims-ms-daemon) *NOTIFY FAILURES* *NOTIFY
DELAYS*

Submitted notifications list:

Address:
#
```

`imta version`

`imta version` prints out the IMTA version number, and displays the system's name, operating system release number and version, and hardware type.

Syntax

```
imta version
```

Example

To check the version of IMTA you are running, execute the following command:

```
% imta version
Sun Internet Mail Server sims.3.5.970720.13
SunOS mailhost 5.5.1 Generic sun4u sparc SUNW, Ultra-1
%
```

Security and Authentication

SIMS 3.5 supports Secure Socket Layers (SSL), version 3.0. SSL encrypts and authenticates messages sent between an Internet Mail Access Protocol version 4 (IMAP4) or Post Office Protocol 3 (POP3) email client and SIMS.

Configuring Secure Socket Layers

Detailed information about SSL and setting up and configuring your SSL security environment is contained in the *Sun Internet Mail Server 3.5 Administrator's Guide*, in the Chapter entitled "Security and Authentication."

Additionally, you can access the following web site for more information about SSL: `http://home.netscape.com/assist/security`. The following section describes how to modify the SSL Key Package, using command-line utilities, after you have set up your SSL environment. Before using these commands, refer to Chapter 7 of the *Sun Internet Mail Server 3.5 Administrator's Guide* in the sections entitled "Creating the SSL Environment," and "Creating a Self-Signed Certificate."

▼ To Delete a Key Package

1. **As** `root`**, type:**

```
# keypkg -D -h -L ip_address
```

This command deletes the server's key package and certificate from the naming service.

▼ To Delete the Root CA Key Package

1. **As** `root` **on the local root CA machine, type:**

```
# keypkg -D -k skirca
```

Note – If you delete the local `root` CA key package, any certificates signed by this local `root` CA will no longer work.

▼ To Change a Key Pair or Distinguished Name for a Certificate

You cannot change the key pair or DN for an existing certificate. If you want to change either of these, you will need to delete the entire key package and then regenerate it.

▼ To Change a Password for a Key Package

- **As** `root`, **type:**

```
# keypkg -P -h -L ip_address
```

You need to enter the old password for the key package, then the new password twice.

▼ To Change the Local Root CA's Password

1. **Log in as user** `skirca`**.**
2. **As** `skirca`**, type:**

```
% keypkg -P
```

You need to enter the old password for the key package, then the new password twice.

Installation

This section describes the utilities that are associated with the installation process. For more information on installation, refer to the *Sun Internet Mail Server 3.5 Advanced Installation Guide*.

setup-tty

`setup-tty` is a script that installs SIMS and related files and packages onto the system.

Note – Because `setup-tty` is not installed on the target system, you must retrieve the `setup-tty` program from the distribution image and not from the system. On the CD, the `setup-tty` script can be found in `/cdrom/sun_internet_mail_3_5/products/sims/setup-tty`.

`setup-tty`'s interface is considered to be "unstable." See `attributes`(5) for a description of interface stability.

Syntax

```
setup-tty [-c install | remove] [-d]
```

The options for this command are:

`-c install`	Specifies a standard install of SIMS and related files and packages.
`-c remove`	Specifies an uninstall of SIMS and related packages and files from the system.
`-d`	Specifies a non-interactive automated install using the `/tmp/sims_setup.dat` file, if it exists. If `/tmp/sims_setup.dat` does not exist, `setup-tty` will default to the standard interactive install and prompt the user for necessary information. See TABLE 2-1 for a description of the parameters included in the `sims_setup.dat` file.

Examples

The following command performs a standard interactive installation:

```
% setup-tty -c install
```

Execute the following to uninstall SIMS and related packages and files from the system:

```
% setup-tty -c remove
```

The following command:

```
% setup-tty -d
```

performs a non-interactive install which uses the file `/tmp/sims_setup.dat` if it exists. It will gather all necessary configuration data from the `/tmp/sims_setup.dat` file. If the file does not exist, `setup-tty` reverts to the interactive install, which prompts the user for necessary information. If `/tmp/sims_setup.dat` exists and `setup-tty` is executed without the `-d` option specified, the `/tmp/sims_setup.dat` file is removed and the interactive install continues

sims_setup.dat File

The following table describes the parameters included in the sims_setup.dat file. The sims_setup.dat file can be provided by the user when the -d option is specified in the setup-tty command.

TABLE 2-1 sims_setup.dat File

Parameter	Description
do_upgrade	Specifies whether or not to upgrade to SIMS 3.5 (1=upgrade, 0=do not upgrade).
upgrade-possible	Specifies whether the current system is upgradable and which SIMS product package it is (0=not upgradable, 1=upgradable departmental, 2=upgradable enterprise).
install-mode	Specifies which SIMS product package to upgrade (1=Departmental, 2=Enterprise, 3=add-on / standalone).
remote-ldap	Determines where the LDAP server is located (0=local host, 1=not local).
calendar	Determines whether or not to install the calendar option (0=do not install, 1=install).
webaccess	Determines whether or not to install the WebAccess option (0=do not install, 1=install).
hjviews	Determines whether or not to install the HotJava Views option (0=do not install, 1=install).
ha_install	Determines whether or not to install the High Availability option. (0=do not install, 1=install).
ha_master	The logical hostname of the HA master host.
ccmail	Determines whether or not to install the cc:Mail channel option (0=do not install, 1=install).
msmail	Determines whether or not to install the Microsoft Mail channel option (0=do not install, 1=install).
profs	Determines whether or not to install the PROFS channel option (0=do not install, 1=install).
remadmin	Determines whether or not to install the remote administrator option (0=do not install, 1=install).
standalone	Determines whether or not SIMS is already installed (0=SIMS already installed, 1=SIMS not installed).
sdk	Determines whether or not to install the SIMS SDK (0=do not install, 1=install).

TABLE 2-1 `sims_setup.dat` File

Parameter	Description
`sdk-doc`	Determines whether or not to install the documentation for the SIMS SDK (0=do not install, 1=install).
`select-options`	Specifies whether or not any options have been selected to install (0=no options selected, 1=HA option only, 2=at least one option selected).
`readfromfile`	Determines whether or not the directory server license will be read from a local file (0=no, 1=yes).
`filename`	Name of the file that contains the SIMS license.
`hostname`	Name of the local host that contains the directory server license.
`redundantservers`	Name of the redundant license server.
`install-lic`	Determines whether or not to install the directory server license (0=do not install, 1=install).
`install-sws`	Determines whether or not to install the Sun web server (0=do not install, 1=install).
`maildomain`	The mail domain.
`rootdomain`	The root domain.
`mta-role`	Determines if the IMTA is installed behind the firewall (0=not behind firewall, 1=behind firewall).
`varmail`	Determines whether or not the `/var/mail` message store is supported (0=not supported, 1=supported).
`postmaster`	The postmaster's ID.
`postmaster_uid`	The postmaster's UID.
`country`	The country code.
`org-name`	The organization code.
`smarthost`	Has a text string value only if `mta-role=1` (behind the firewall).
`ou`*x*	The descriptive name for the organizational unit. *x* is a numerical value starting with 1 and continuing up 6 units.
`ou`*x*`-domain`	The domain for the organizational unit. *x* is a numerical value starting with 1 and continuing up to 6 units.
`province`	The name of the province. This is a text entry.
`locality`	The name of the locality. This is a text entry.

TABLE 2-1 `sims_setup.dat` File

Parameter	Description
`postal-address`	The postal address. This is a text entry.
`phone-number`	The phone number. This is a text entry.
`fax-number`	The fax number. This is a text entry.
`org-name-long`	The name of the organization. This is a text entry.
`ldap-server`	Hostname of the LDAP master server.
`ldap-port`	The LDAP port number.
`mmad-port`	The MMAD port number.
`smcs-domain`	The domain name of the Sun Messaging Connectivity Services (SMCS) server.
`smcs-domain-dns`	The name of the DNS for SMCS.
`smtp-port`	The SMTP port number.
`alias-format`	Alias format of the SMCS-IN channel.
`alias-format-str`	The user-entered alias format string.
`document-root`	The location of the document root for the Sun Web Server.
`cgi-bin`	The location of the CGI bin directory for the HotJava Views server.
`ha_masterlhost`	The logical host name for the HA installation.
`ha_sharedfs`	The shared disk location for the HA installation.
`administrator-name`	The user name for the administrator.
`administrator-passwd`	The password for the administrator.

uninstall

The `uninstall` utility removes SIMS and other related files and packages from your system. You can specify `uninstall` to perform a standard or dramatic procedure.

Note – `uninstall` may not remove certain packages that are likely to have been installed by a separate application and may be used by that application. This is the case even if SIMS has installed that package upon setup.

`sendmail` is restored by the SIMS `uninstall` utility but it is not started. In order to start `sendmail`, the user must either reboot the system or manually start the `sendmail` program.

Web server packages are removed by `uninstall`, but `httpd` is not stopped.

Syntax

```
uninstall [-c sims] [-d sims]
```

The options for this command are:

`-c sims`	Specifies a standard `uninstall` of SIMS and related files and packages. The standard `uninstall` does not remove the directory or the message store. Only the binaries are removed.
`-d sims`	Specifies a dramatic `uninstall` of SIMS and related files and packages. This option removes data and configuration files left over from the standard `uninstall`. The dramatic `uninstall` option is a clean `uninstall`, removing all files installed by the SIMS installation process and created by SIMS during operation, with the exception of packages that may have already been present before the `uninstall` procedure.

Examples

The following command performs a standard `uninstall`:

```
% uninstall -c sims
```

The following command performs a dramatic `uninstall`:

```
% uninstall -d sims
```

Sun Messaging Connectivity Services

This section describes how to start up the Sun Messaging Connectivity Services (SMCS) to help you appropriately configure global settings for your site's Sun Internet Mail Server (SIMS–the *server*) and channel gateway (the *client*).

Starting the Sun Messaging Connectivity Services

The following information describes how to use the `smcs` start up script. This utility is used to perform a variety of maintenance functions for SMCS. This utility should not be modified. Do not edit the SMCS script.

The `smcs` Utility

The following information describes the arguments and command-line syntax for the `smcs` start up script.

`smcs [-p` *number*`] [-d` *directory*`] backup`

`backup` restricts access to SMCS and performs a copy of essential files into another location. It is recommended to backup to disk first, and then use normal backup procedures. The target directory should be empty because multiple files are backed up and if the directory is not empty you will not know the name of the files.

`smcs delete`

`delete` removes all nonstandard files from an SMCS configuration. It also removes SMCS installed entries from the running directory.

Caution – `smcs delete` removes all SMCS configurations, and should be performed with caution.

smcs [-p *number*] dirsync

dirsync starts a Directory Synchronization based on the current configuration. This command is normally run periodically from cron if the user has scheduled a dirsync from the directory UI.

smcs getport

Returns the MMAD port number.

smcs [-p *number*] initialize

initialize prompts for variables that are used to configure a newly-installed SMCS configuration. The initialize option can be used with new installations or after using smcs delete. The following is a sample initialization script:

```
WARNING: Initialize is a non-reversible, destructive operation!
Are you sure you want to continue? (y/[n]) y
Please enter the SMCS Search Base DN: ou=abc,o=xyz,c=us
Please enter the SMCS Root DN: ou=services,ou=abc,o=xyz,c=us
Please enter the fully qualified local host name: abc.xyz.com
Please enter the IMTA->SMCS SMTP port [27]:
Please enter the Administrator DN: cn=admin,o=xyz,c=us
Please enter the directory Administrator password:
```

smcs [-p *number*] nightly

nightly is added to cron during installation and runs a series of maintenance and informational routines such as clean-up and maintenance of message queues. nightly is run at least once per day from cron. The standard output from this command is mailed to the postmaster (usually root).

The following tasks are executed the smcs nightly script. Create a file in /etc/opt/SUNWmail/gtw/cfg called nightly_opts to change the default options. The nightly_opts file should have an entry for each of the options. The following is a sample nightly_opts file. The default options are listed.

```
NIGHTLY_TIDY = 1 /* Send back undeliverable messages */
NIGHTLY_R_TMP = 1 /* Removes temporary files */
NIGHTLY_PURGE = 0 /* Delete messages older than 3 days */
NIGHTLY_BACKUP = 0 /* Backup and configuration */
```

`smcs [-p` *number*`] purge`

`purge` removes all queued messages with a “Deleted” status, plus any miscellaneous tracing logs.

`smcs [-p` *number*`] restart`

`restart` confirms that the MMA is running, and restarts it if it is idle. `smcs restart` is frequently run from `root`’s `crontab`. By default, `restart` is set to run every 10 minutes.

`smcs [-d` *directory*`] restore`

`restore` performs the complement to the backup operation and should be performed with caution as a `restore` replaces the current configuration. The target directory should be the same directory that was used with the `-d` option in the `smcs backup` utility.

`smcs setport`

Prompts the user for the new MMAD port number. SMCS then sets the MMAD port number accordingly.

`smcs [-p` *number*`] start`

`start` starts the SMCS Management Agent (MMAD) and then attempts to start the Router, the Transports, and the configured channels. The `-p` option is needed *only* if the management agent is running on a port other than the default (2585), which is *not* recommended. `smcs start` is part of the system initialization scripts and is, by default, added to `/etc/inittab` by the installation script.

`smcs [-p` *number*`] stop`

`stop` stops the MMA in a process similar to `smcs start`. You should *only* run this when serious problems occur, such as, the system shuts down, or a restoration is needed.

Client Configuration Applications

config.exe

When the PC client software is installed, `setup.exe` is copied to the specified hard drive with the name `config.exe`. This allows you to modify SMCS client parameters at any time. The setup program creates a windows program group named “PC Clients.”

setup.exe

The `setup.exe` application installs the client.

uninstall.exe

The `uninstall.exe` application removes the client.

cc:Mail Client Applications

ccclient.exe

This program is used to transfer messages between IMTA and cc:Mail. The client must be running for interaction to occur with cc:Mail. The program can be started manually from the DOS prompt, or may be added to a batch file. Commands must be run from the `\CCMAIL\` subdirectory.

To interrupt `CCCLIENT`, press <Esc>. Commands include:

`ccclient`	Runs cc:Mail client continuously. Mail is picked up and delivered according to polling intervals specified in `config.exe`.
`ccclient -1`	Runs each cc:Mail client one time. When this occurs, it picks up and delivers mail that currently exists in specified queues.

`dassist.exe`

This executable imports LDAP entries and exports cc:Mail directories for directory synchronization. `dassist` can be run manually, or via `sched.exe` at intervals specified in `config.exe`. Commands must be run from the `\CCMAIL\DASSIST` subdirectory. To interrupt `dassist`, press <Esc>. Commands include:

`export`	Creates a directory sync export file for a specified cc:Mail channel. These files must be created prior to a directory sync cycle begins via the Directory Sync Management Utility (DSMU) on the server. The command is entered as follows: `dassist` *channel_name* `export`
`import`	Imports a directory sync import file for a specified cc:Mail channel. These files must be created after the directory synchronization utility (`dirsync`) has finished creating the import file on the server. The command is entered as follows: `dassist` *channel_name* `import`

`sched.exe`

This is the directory sync schedule utility used for cc:Mail. The program can be started manually from the DOS prompt, or can be added to a batch file. Commands must be run from the \CCMAIL\DASSIST subdirectory. To interrupt `sched.exe`, press Ctrl-C.

Microsoft Mail Client Applications

`msclient.exe`

This program is used to transfer messages between IMTA and Microsoft Mail. The client must be running for IMTA to interact with Microsoft Mail. The program can be started manually from the DOS prompt, or may be added to a batch file. Commands must be run from the `\MSMAIL\` subdirectory. To interrupt `msclient`, press <Esc>.

Commands include:

`msclient`	Runs ’s Microsoft Mail client continuously. Mail is picked up and delivered according to polling intervals specified in `config.exe`.
`msclient -1`	Runs ’s Microsoft Mail client one time. When this occurs, it picks up and delivers mail that currently exists in specified queues.

msvexp.exe

Imports and exports Microsoft Mail directories for directory synchronization. `msvexp.exe` can be run manually, or via `mssched.exe` at intervals specified. Commands must be run from the `\MSMAIL\DIRSYNC` subdirectory. To interrupt `msvexp.exe`, press Ctrl-C.

Commands include:

`msvexp /e`	Creates a directory sync export file. These files must be created prior to when a directory sync cycle begins via the Directory Sync Management Utility (DSMU) on the server.
`msvexp /i`	Imports a directory sync import message. These messages must be created after the Directory Sync Management Utility (DSMU) on the server has finished creating the import file.

mssched.exe

This is the directory sync schedule utility used by the Microsoft Mail client. The program can be started manually from the DOS prompt, or may be added to a batch file. Commands must be run from the `\MSMAIL\DIRSYNC` subdirectory. To interrupt `mssched.exe`, press Ctrl-C.

When running the program manually, you are prompted to enter export and import times manually using an hh:mm format.

PROFS Client Applications

`njecmd.exe`

This is an OS/2 administrative utility used for issuing commands to the `njeServer` daemon.

`njeserv.exe`

This is the NJE daemon used as communication between PROFS and SIMS for OS/2. `njeserv.exe` runs as a daemon and sends and receives messages from PROFS.

CHAPTER **3**

IMTA Configuration

Five major Internet Message Transfer Agent (IMTA) configuration files are supported by Sun Internet Mail Server 3.5 to be editable with any text editor.

- `imta.cnf`
- `mappings`
- `option.dat`
- `job_controller.cnf`
- `dispatcher.cnf`

Configuration modifications can be done using the command-line interface, as described in this manual, or by using the accompanying Graphical User Interface (GUI), as described in the *Sun Internet Mail Server 3.5 System Administrator's Guide*.

This section explains how to modify the IMTA configuration files to obtain behaviors not directly configurable from the Java administration front end. Sun recommends that only experienced administrators edit and modify the configuration files. Configuration settings can be defined using the GUI described in the *Sun Internet Mail Server System Administrator's Guide*. The command-line options for configuring your system are described in the following sections and in Chapter 2, "Commands Reference."

Caution – Sun does not guarantee that the changes made by modifying configuration files will be recorded properly by the administration console.

Note – If you do change any of the files manually, be sure to restart your administration server after you make the changes. Then restart the administration console. This will ensure that the information that you changed in the `imta.cnf` file get synchronized in the administration console.

All configuration files are ASCII text files that can be created or changed with any text editor. Permissions for the configuration file should be set to world-readable. Failure to make configuration files world-readable may cause unexpected IMTA failures. A physical line in the files is limited to 252 characters. You can split a logical line into multiple physical lines using the backslash (\) continuation character.

By pre-processing these files and storing them into an image, the initialization time for IMTA is significantly reduced, thereby improving IMTA's performance.

`imta.cnf` Configuration File

The `imta.cnf` file contains the routing and address rewriting configuration. It defines all channels and their characteristics, the rules to route mail among those channels, and the method in which addresses are rewritten by the IMTA.

Structure of the `imta.cnf` File

The configuration file consists of two parts: domain rewriting rules and channel definitions. The domain rewriting rules appear first in the file and are separated from the channel definitions by a blank line. The channel definitions are collectively referred to as the channel table. Individual channel definitions form a channel block.

Comments in the File

Comment lines may appear anywhere in the configuration file. A comment is introduced with an exclamation point (!) in column one. Liberal use of comments to explain what is going on is strongly encouraged. The following `imta.cnf` file fragment displays the use of comment lines.

```
! Part I: Rewrite rules
!
sims-ms.my_server.my_company.com $E$U@sims-ms-daemon
!
! Part II: Channel definitions
```

Distinguishing between blank lines and comment lines is important. Blank lines play an important role in delimiting sections of the configuration file. Comment lines are ignored by the configuration file reading routines—they are literally “not there” as far as the routines are concerned and do not count as blank lines.

Including Other Files

The contents of other files may be included in the configuration file. If a line is encountered with a less than sign (<) in column one, the rest of the line is treated as a file name; the file name should always be an absolute and full file path. The file is opened and its contents are spliced into the configuration file at that point. Include files may be nested up to three levels deep. The following `imta.cnf` file fragment includes the `/etc/opt/SUNWmail/table/internet.rules` file.

```
</etc/opt/SUNWmail/table/internet.rules
```

Note – Any files included in the configuration file must be world-readable just as the configuration file is world-readable.

Channels

The central unifying construct in IMTA is the channel. A *channel* is some form of connection with another system or group of systems. Here, *system* is being used quite loosely and may mean another computer system, mail system, user agent, or gateway. The actual hardware connection or software transfer, or both, may vary widely from one channel to the next. Only the IMTA manager needs to know anything about IMTA’s channels. Users are never aware of the existence of channels and only see a single, uniform interface regardless of how messages reach their destination.

Each channel consists of one or more channel programs and an outgoing message queue for storing messages that are destined to be sent to one or more of the systems associated with the channel. Channel programs perform two functions:

- Transmit messages to other systems, deleting them from their queue after they are sent
- Accept messages from other systems, placing (enqueuing) them into channel queues

Note – While a channel program only removes messages from its own queue, it can enqueue messages to any queue, including its own.

A channel program that initiates a transfer to another system or another channel on its own is called a "master" program, while a program that accepts transfers initiated by another system is called a "slave" program. A channel may be served by a master program, a slave program, or both. Either type of program may or may not be bidirectional; the direction in which a message is traveling may have nothing to do with the type of program that handles it.

Channel Queue Formats

Messages queued for delivery are always stored in the same format, regardless of the type of channel in which they are queued. All messages are stored in subdirectories under the IMTA queue directory. There is usually one subdirectory per channel; its name is the same as the channel name. For example, messages queued for delivery in the Sun Message Store (sims-ms) channel are stored in the `/var/opt/SUNWmail/imta/queue/sims-ms` directory. Each file contains a single message.

Note – Some temporary files are stored in the top-level queue directory. The names of these temporary files usually begin with a dollar sign (`$`).

All of these directories are protected against access by non-privileged users. The first two characters of each file name are a representation of the number of times delivery has been attempted on the file. This information is encoded in "complemented base 36." For example, if no attempts have been made to deliver a message, the file name will begin with ZZ. The name will begin with ZY if one attempt has been made, ZA if 25 attempts have been made, Z0 if 35 attempts have been made, YZ if 36 attempts have been made, and so on.

The remainder of the file names are pseudo-random strings of hexadecimal characters that serve to make the file names unique. The file type (suffix) is always a pair of letters or digits, usually `00`. Messages being held have `.HELD` as the file type; these messages are not eligible for delivery processing. Examine the `.HELD` files and correct the reason they could not be delivered. Then use `imta queue-retry-delivery` *channel-name* to deliver the `.HELD` messages.

The actual internal format of the message files is irrelevant. Those who wish to write their own IMTA channel programs, should access the messages via the documented IMTA API interface. However, it is sometimes useful for a system manager to examine messages in the queues, so it is helpful to note that messages are stored as ASCII text and message files may be typed on a terminal without adverse effects.

Message Structure

Most IMTA messages are stored as text files. Messages with multiple parts (possibly containing different types of data) are represented as a series of text sections separated by special unique delimiter strings.

A sample mail message file is given below.

```
m;DAN@SIGURD.COMPANY.COM  (1)
ned@YMIR.UNIVERSITY.EDU
(2)
Received: from SIGURD.COMPANY.COM by SIGURD.COMPANY.COM (IMTA
#11000) id
<01G6YTYFU6748WWH0Y@SIGURD.COMPANY.COM>; Thu, 13 Jun 1991 15:01
PDT
Date: Thu, 13 Jun 1991 15:01 PDT
From: "Daniel C. Newman" <DAN@SIGURD.COMPANY.COM>
Subject: Mrochek ate my shoe
To: ned@YMIR.UNIVERSITY.EDU
Message-id: <01G6YTYFU6748WWH0Y@SIGURD.COMPANY.COM>
MIME-version: 1.0
Content-type: TEXT/PLAIN; CHARSET=US-ASCII
Content-transfer-encoding: 7BIT
X-Envelope-to: ned@YMIR.UNIVERSITY.EDU
X-VMS-To: IN%"ned@ymir.university.edu"
(3)
Ned,

             Mrochek ate another shoe of mine....

             Dan

              (4)
(5)
```

Briefly, the key items in each message file are:

1. The message envelope. The first records in the file contains message envelope (i.e., transport) information. The envelope is terminated by a record containing to two CTRL/A characters.
2. The header lines of the message follow the envelope; their format is mandated by RFC 822.

3. There may be any number of message header lines; the message header formed by this collection of header lines is terminated by a single blank line after which follows the message body.
4. The message is terminated by a sequence of five CTRL/As.
5. For messages that had transient delivery failures, information about retrial would go here.

Note – Sun reserves the right to change this format in future releases of the IMTA. User-written applications that either read or write queued IMTA message files should make use of appropriate IMTA library routines. Use of the IMTA SDK will insulate user applications from any future message format changes.

Domain Rewriting Rules

Domain rewriting rules, or, as they are frequently called, “rewrite rules,” play two important roles. First, they are used to rewrite addresses into their proper or desired form. Second, they are used to determine to which channels a message should be enqueued. The determination of which channels to enqueue a message is made by rewriting its envelope `To: addresses`. Each rewrite rule appears on a single line in the upper half of the `imta.cnf` file.

For additional information about the domain rewriting process, refer to the *Sun Internet Mail Server 3.5 Administrator’s Guide*.

Rewriting Rules Structure

The rewrite rules appear in the upper-half of the IMTA configuration file, `imta.cnf` (see the sample configuration file in the following section, “Configuration File Format”). Each rule in the configuration file appears on a single line. Comments, but not blank lines, are allowed between the rules. The rewrite rules end with a blank line, after which the channel definitions follow.

Rewrite rules consist of two parts: a pattern followed by an equivalence string or “template.” The two parts must be separated by spaces, although spaces are not allowed within the parts, themselves. The template specifies a *username*, any applicable *options*, a *host/domain* specification, and the name of a system attached to an existing IMTA channel (the *routing system*), to which messages to this address are sent. The structure for rewriting rules is:

```
pattern controls [userTemplate]%[domainTemplate]@[routingSystem] controls
```

The following table describes the parts of the rewriting rule structure:

`pattern`	The rule applies if the pattern is a substring of the domain part of the address. An asterisk (*) can be used as wild card character.
`controls`	The applicability of a rule can be limited using these control sequences. Control sequences may be located either before the user template or after the routing system. The selection criteria are described in TABLE 3-5. They include: • Envelope or header addresses • direction (To or From) • destination channel of the message Or, the following contexts: • Addresses in messages directed to a specified or unspecified channel
[*userTemplate*]	Specifies how the user part of the address is rewritten. The template may be built using substitution sequences to represent certain parts of the original address or the results of a database lookup. The substitution sequences are replaced with what they represent in order to construct the rewritten address. See TABLE 3-3.
%	Separator used between *userTemplate* and *domainTemplate* (see preceding structure sample).
[*domainTemplate*]	Specifies how the domain part of the address is rewritten. Like the *userTemplate*, the *domainTemplate* may be built using substitution sequences.
@	Separator used between *domainTemplate* and *routingSystem* (see preceding structure sample).
[*routingSystem*]	Specifies the destination channel's routing system. Every channel is associated with a string (the *routingSystem*).

Refer to the section, "Template Substitutions" for additional information about rewrite rule structures and concepts.

Rewriting Rules Operation

The following steps apply to the application of the domain rewriting rules to a given address:

1. The first host or domain specification is extracted from an address. An address may specify more than one host or domain name as in the case:

   `jdoe%hostname@alpha.com`.

2. After identifying the first *host* or *domain* name, a search is conducted that scans for a rewrite rule whose pattern matches the host/domain name.

3. When the matching rewrite rule is found, the address is rewritten according to the template portion of that rule. The template also specifies the name of a routing system to which messages sent to this address are routed. (In this case, the term “routing system” does not necessarily mean the name of a system through which the message is routed, but rather a tag associated with a specific channel.)

4. Finally, the routing system name is compared with the host names that are associated with each channel. If a match is found, the message is enqueued to that channel; otherwise, the rewriting process fails. If the matching channel is the local channel, some additional rewriting of the address may take place by looking up the aliases database because the local channel rules are used to identify any local users as well as any `/var/mail` users.

Note – Using a routing system that does not belong to any existing channel will cause messages whose addresses match this rule to be bounced. That is, it makes matching messages non-routable.

Extracting the First Host/Domain Specification

The process of rewriting an address starts by extracting the first host/domain specification from the address. (Readers who are not familiar with RFC 822 address conventions are advised to read that standard, at least in a cursory fashion, at this point in order to understand the following discussion.) The order in which host/domain specifications in the address are scanned is as follows:

- Hosts in source routes (read from left to right)
- Hosts appearing to the right of the at sign (`@`)
- Hosts appearing to the right of the last single percent sign (`%`)
- Hosts appearing to the left of the first exclamation point (`!`)

The order of the last two items are switched if the `bangoverpercent` keyword is in effect on the channel that is doing the address rewriting. That is, if the channel attempting to enqueue the message is, itself, marked with the `bangoverpercent` channel keyword.

Some hypothetical examples of addresses and the hostnames that could be extracted first are shown in TABLE 3-1:

TABLE 3-1 Examples of Extracted Addresses and Hostnames

Address	First host domain specification	Comments
user@a	a	a is a "short-form" domain name
user@a.b.c	a.b.c	a.b.c is a "fully qualified" domain name (FQDN)
user@[0.1.2.3]	[0.1.2.3]	[0.1.2.3] is a "domain literal"
@a:user@b.c.d	a	This is a source-routed address with a short-form domain name, the "route"
@a.b.c:user@d.e.f	a.b.c	Source-routed address; route part is fully-qualified
@[0.1.2.3]:user@d.e.f	[0.1.2.3]	Source-routed address; route part is a domain literal
@a,@b,@c:user@d.e.f	a	Source-routed address with an a to b to c routing
@a,@[0.1.2.3]:user@b	a	Source-routed address with a domain literal in the route part
user%A@B	B	This nonstandard form of routing is called a "percent hack"
user%A%B%C@D	D	A built-up percent hack
user%A	A	
user%A%B	B	
user%%A%B	B	
A!user	A	"Bang-style" addressing; commonly used for UUCP
A!user@B	B	
A!user%B@C	C	
A!user%B	B	`nobangoverpercent` keyword active; the default
A!user%B	A	`bangoverpercent` keyword active

RFC 822 does not address the interpretation of exclamation points (!) and percent signs (%) in addresses. Percent signs are customarily interpreted in the same manner as at signs (@) if no at sign is present, so this convention is adopted by IMTA.

The special interpretation of repeated percent signs is used to allow percent signs as part of local usernames; thus is used in handling PSIMail and other foreign mail system addresses. The interpretation of exclamation points conforms to RFC 976's "bang-style" address conventions and makes it possible to use UUCP addresses with IMTA.

The order of these interpretations is not specified by either RFC 822 or RFC 976, so the `bangoverpercent` and `nobangoverpercent` keywords can be used to control the order in which they are applied by the channel doing the rewriting. The default is more "standard," although the alternate setting may be useful under some circumstances.

Note – The use of exclamation points (!) or percent signs (%) in addresses is not recommended. It is preferable to convert them into regular internet addresses using the patterns $! or $%.

Scanning the Rewrite Rules

Once the first host/domain specification has been extracted from the address, the IMTA consults the rewrite rules to find out what to do with it. The host/domain specification is compared with the pattern part of each rule (i.e., the left-hand side of each rule). The comparison is case insensitive. Case insensitivity is mandated by RFC 822, UUCP addresses notwithstanding. The IMTA is insensitive to case but preserves it whenever possible.

If the host/domain specification does not match any pattern, in which case it is said to "not match any rule", the first part of the host/domain specification—the part before the first period, usually the host name—is removed and replaced with an asterisk and another attempt is made to locate the resulting host/domain specification, but only in the configuration file rewrite rules (the domain database is not consulted). If this fails the first part is removed and the process is repeated. If this also fails the next part is removed (usually a subdomain) and the rewriter tries again, first with asterisks and then without. This process proceeds until either a match is found or the entire host/domain specification is exhausted. The effect of this procedure is to try to match the most specific domain first, working outward to less specific and more general domains.

A somewhat more algorithmic view of this matching procedure is:

- The host/domain specification is used as the initial value for the comparison strings `spec_1` and `spec_2`. (For example, `spec_1 = spec_2 = a.b.c`).
- The comparison string `spec_1` is compared with the pattern part of each rewrite rule in the configuration file and then the domain database until a match is found. The matching procedure is exited if a match is found.

- If no match is found then the left-most, non-asterisk part of spec_2 is converted to an asterisk. For example, if spec_2 is a.b.c then it is changed to *.b.c; if spec_2 is *.b.c then it is changed to *.*.c. The matching procedure is exited if a match is found.
- If no match is found then the first part, including any leading period, of the comparison string spec_1 is removed. In the case where spec_1 has only one part (for example., .c or c), the string is replaced with a single period, ".". If the resulting string spec_1 is of non-zero length, then we return to Step 1. If the resulting string has zero length (for example, was previously ".") then the lookup process has failed and we exit the matching procedure.

For example, suppose the address dan@sc.cs.cmu.edu is to be rewritten. This causes the rewriter to look for the following patterns in the given order:

```
sc.cs.cmu.edu
*.cs.cmu.edu
.cs.cmu.edu
*.*.cmu.edu
.cmu.edu
*.*.*.edu
.edu
*.*.*.*
.
```

Testing Domain Rewriting Rules

You can test rewrite rules with the `imta test -rewrite` command. The `-noimage` qualifier will allow you to test changes made to the configuration file prior to recompiling and reinstalling the new configuration.

You may find it helpful to rewrite a few addresses using this utility with the `-debug` qualifier. This will show you step by step how the address is rewritten. For instance, try issuing the command:

```
% imta test -rewrite joe@alpha.com
```

and see what happens. For a detailed description of the `imta test -rewrite` utility, refer to Chapter 2, "Commands Reference."

Simple Configuration File

The following example of an `imta.cnf` configuration file shows how rewrite rules are used to route messages to the proper channel. No domain names are used in order to keep things as simple as possible.

```
! test.cnf - An example configuration file. (1)
!
! This is only an example of a configuration file. It serves
! no useful purpose and should not be used in a real system.
!
a     $U@a-daemon (2)
b     $U@b-daemon
c     $U%c@b-daemon
d     $U%d@a-daemon
              (3)
l             (4)
local-host

a_channel defragment charset7 usascii (5)
a-daemon

b_channel noreverse notices 1 2 3
b-daemon
```

The key items (labeled with boldface numbers, enclosed in parentheses) in the preceding configuration file are explained in the following list:

1. Exclamation points (!) are used to include comment lines. The exclamation point must appear in the first column. An exclamation point appearing anywhere else is interpreted as a *literal* exclamation point.
2. The rewrite rules appear in the first half of the configuration file. Absolutely no blank lines should appear among the lines of rewrite rules. Lines with comments (beginning with an exclamation point in the first column) are, however, permitted.
3. The first blank line to appear in the file signifies the end of the rewrite rules section and the start of the channel blocks.
4. The first channel block to appear is always channel l (the local channel, designated with the lowercase letter “l”). Blank lines then separate each channel block from one another. An exception is a defaults channel, which can appear before channel l.
5. A channel named `a_channel`. Notice the use of a channel keyword (`822`) with this channel.

TABLE 3-2 lists the routing and queuing of messages by the preceding configuration:

TABLE 3-2 Address Routing and Channel Queuing

Address	Queued to channel
u@a	a_channel
u@b	b_channel
u@c	b_channel
u@d	a_channel
u@e	a_channel
u@f	b_channel

Template Substitutions

Substitutions are used to abbreviate user names or addresses by inserting a character string into the rewritten address, the value of which is determined by the particular substitution sequence used. For example, in the template:

```
$U@acme.com
```

the `$U` is a substitution sequence. It causes the *username* portion of the address being rewritten to be substituted into the output of the template. Thus, if `jdoe@mailhost.acme.com` was being rewritten by this template, the resulting output would be `jdoe@acme.com`, the `$U` substituting in the *username* portion, `jdoe`, of the original address.

A summary of template substitutions is contained in TABLE 3-3.

TABLE 3-3 Summary of Template Substitutions

Substitution Sequence	Substitutes
$D	Portion of domain specification that matched
$H	Unmatched portion of host/domain specification; left of dot in pattern
$L	Unmatched portion of domain literal; right of dot in pattern literal
$U	Username from original address
$$	Inserts a literal dollar sign ($)
$%	Inserts a literal percent sign (%)

TABLE 3-3 Summary of Template Substitutions

Substitution Sequence	Substitutes
$@	Inserts a literal at sign (@)
$\	Force material to lowercase
$^	Force material to uppercase
$_	Use original case
$W	Substitutes in a random, unique string
$[...]	Invoke customer-supplied routine; substitute in result
$(text)	General database substitution; rule fails if lookup fails
${...}	Apply specified mapping to supplied string
$&n	nth part of unmatched (or wild card) host, as counting from left to right, starting from 0
$!n	nth part of unmatched (wild card) host, as counted from right to left, starting from 0
$*n	nth part of matching pattern, as counting from left to right, starting from 0
$#n	nth part of matching pattern, as counted from right to left, starting from 0

Customer-Supplied Routine Substitutions, `$[...]`

A substitution of the form $[*image,routine,argument*] is handled specially. The *image,routine,argument* part is used to find and call a customer-supplied routine. At runtime, IMTA uses `dlopen` and `dlsym` to dynamically load and call the routine *routine* from the shared library image. The routine *routine* is then called as a function, with the following argument list:

```
int routine    (char *argument     /* input string */
               int *arglength      /* pointer to length of input
                                   string
               char *result        /* result of substitution */
               init *reslength);   /* length of result of
                                   substitution */
```

`argument` and `result` are 252 byte-long-character string buffers. The routine *routine* returns a `0` if the rewrite rule fails, and `-1` if the rewrite rule succeeds.

This mechanism allows IMTA's rewriting process to be extended in complex ways. For example, a call to a name service could be performed, and the result used to alter the address. For example, directory service lookups for forward-pointing addresses (`To:` addresses) to the host `alpha.com` might be performed as follows, with the rewrite rule, `$F`, described in TABLE 3-5 causing this rule to be used only for forward-pointing addresses):

```
jdoe@acme.com $F$[libxyz.so,mylookup,$U]
```

A forward-pointing address, `jdoe@acme.com`, when it matches this rewrite rule, causes `libxyz.so` to be loaded into memory, then causes the routine `mylookup` called with `jdoe` as the argument parameter. The routine `mylookup` might then return a different string, say, `John.Doe%alpha.com` in the `result` parameter and the value `-1` to indicate that the rewrite rule succeeded. The percent sign (%) in the `result` string causes the rewriting process to start over again, using `John.Doe@alpha.com` as the address to be rewritten. The site-supplied shared library image *image* should be world readable.

Note – This facility is not designed for use by casual users; it is intended to be used to extend IMTA's capabilities system-wide.

Source Channel-Specific Rewrite Rules (`$M, $N`)

Rewrite rules can possibly act only in conjunction with specific source channels. This is useful when a short-form name has two meanings:

1. When it appears in a message arriving on one channel
2. When it appears in a message arriving on a different channel

Source channel-specific rewriting is associated with the channel program in use and the channel keywords `rules` and `norules`. If `norules` is specified on the channel associated with an IMTA component that is doing the rewriting, no channel-specific rewrite checking is done. If `rules` is specified on the channel, then channel-specific rule checks are enforced. `rules` is the default.

Source channel-specific rewriting is not associated with the channel that matches a given address. It depends only on the IMTA component doing the rewriting and that component's channel table entry. Channel-specific rewrite checking is triggered by the presence of a `$N` or `$M` control sequence in the template part of a rule. The characters following the `$N` or `$M`, up until either an at sign (@), percent sign (%), or subsequent `$N`, `$M`, `$Q`, `$C`, `$T`, or `$?` are interpreted as a channel name.

`$M` *channel* causes the rule to fail if the channel *channel* is not currently doing the rewriting. `$N` *channel* causes the rule to fail if the channel *channel* is doing the rewriting. Multiple `$M` and `$N` clauses may be specified. If any one of multiple `$M` clauses matches, the rule succeeds. If any of multiple `$N` clauses matches, the rules will fail.

Destination Channel-Specific Rewrite Rules (`$C`, `$Q`)

Rewrite rules can possibly act only in conjunction with the channel to which the message is being queued. This is useful a host has two names, one known to one group of hosts and one known to another. By using different channels to send mail to each group, addresses can be rewritten to refer to the host under the name known to each group.

Destination channel-specific rewriting is associated with the channel to which the message is to be dequeued and processed by, and the channel keywords `rules` and `norules` on that channel. If `norules` is specified on the destination channel, no channel-specific rewrite checking is done. If `rules` is specified on the destination channel, channel-specific rule checks are enforced. `rules` is the default.

Destination channel-specific rewriting is not associated with the channel matched by a given address. It depends only on the message's envelope `To:` address. When a message is enqueued, its envelope `To:` address is first rewritten to determine to which channel the message is enqueued. During the rewriting of the envelope `To:` address, any `$C` and `$Q` control sequences are ignored. After the envelope `To:` address is rewritten and the destination channel determined, then the `$C` and `$Q` control sequences are honored, as other addresses associated with the message are rewritten.

Destination channel-specific rewrite checking is triggered by the presence of a `$C` or `$Q` control sequence in the template part of a rule. The characters following the `$C` or `$Q`, up until either an at sign (`@`), percent sign (`%`), or subsequent `$N`, `$M`, `$C`, `$Q`, `$T`, or `$?` are interpreted as a channel name.

`$Q` *channel* causes the rule to fail if the channel *channel* is not the destination. `$C` *channel* causes the rule to fail if the channel *channel* is the destination. Multiple `$Q` and `$C` clauses may be specified. If any one of multiple `$Q` clauses matches, the rule succeeds. If any of multiple `$C` clauses matches, the rule fails.

Direction- and Location-Specific Rewrites ($B, $E, $F, $R)

Sometimes you need to specify rewrite rules that apply only to envelope addresses or, alternately, only to header addresses. The control sequence `$E` forces a rewrite to fail if the address being rewritten is not an envelope address. The control sequence `$B` forces a rewrite to fail if the address being rewritten is not from the message header or body. These sequences have no other effects on the rewrite and may appear anywhere in the rewrite rule template.

Addresses may also be categorized by direction. A forward-pointing address is one that originates on a `To:`, `Cc:`, `Resent-to:`, or other header or envelope line that refers to a destination. A backward-pointing address is something like a `From:`, `Sender:`, or `Resent-From:`, that refer to a source. The control sequence `$F` causes the rewrite to fail if the address is backward-pointing. The control sequence `$R` causes the rewrite to fail if the address is forward-pointing.

Host Location-Specific Rewrites ($A, $P, $S, $X)

Circumstances occasionally require rewriting that's sensitive to the location where a host name appears in an address. Host names can appear in several different contexts in an address:

- In a source route
- To the right of the at sign (@)
- To the right of a percent sign (%) in the local-part
- To the left of an exclamation point in the local-part

Under normal circumstances, a host name should be handled in the same way, regardless of where it appears. Situations can arise, however, that may necessitate specialized handling.

Four control sequences are used to control matching on the basis of the host's location in the address.

1. `$S` specifies that the rule can match a host extracted from a source route.
2. `$A` specifies that the rule can match a host found to the right of the `@` sign.
3. `$P` specifies that the rule can match a host found to the right of a `%` sign.
4. `$X` specifies that the rule can match a host found to the left of an exclamation point (!).

The rule fails if the host is from a location other than one specified. These sequences can be combined in a single rewrite rule. For example, if $S and $A are specified, the rule matches hosts specified in either a source route or to the right of the at sign. Specifying none of these sequences is equivalent to specifying all of them; the rule can match regardless of location.

Single Field Substitutions ($&, $!, $*, $#)

Single field substitutions extract a single subdomain part from the host/domain specification being rewritten. The available single field substitutions are shown in TABLE 3-4.

TABLE 3-4 Single Field Substitutions

Control Sequence	Usage
`$&n`	Substitute the nth element, n=0,1,2,..,9, in the host specification (the part that did not match or matched a wildcard of some kind). Elements are separated by dots; the first element on the left is element zero. The rewrite fails if the requested element does not exist.
`$!n`	Substitute the nth element, n=0,1,2,..,9, in the host specification (the part that did not match or matched a wildcard of some kind). Elements are separated by dots; the first element on the right is element zero. The rewrite fails if the requested element does not exist.
`$*n`	Substitute the nth element, n=0,1,2,...,9, in the domain specification (the part that did match explicit text in the pattern). Elements are separated by dots; the first element on the left is element zero. The rewrite fails if the requested element does not exist.
`$#n`	Substitute the nth element, n=0,1,2,...,9, in the domain specification (the part that did match explicit text in the pattern). Elements are separated by dots; the first element on the right is element zero. The rewrite fails if the requested element does not exist.

Suppose the address `jdoe@vaxa.acme.com` matches the following rewrite rule:

```
*.ACME.COM     $U%$&0.acme.com@mailhub.acme.com
```

Then the result from the template will be `jdoe@vaxa.acme.com` with `mailhub.acme.com` used as the routing system.

Handling Domain Literals

Domain literals are handled specially during the rewriting process. If a domain literal appearing in the left of an address does not match, the literal is interpreted as a group of strings separated by periods and surrounded by square brackets.6 The right-most string is removed and the search is repeated. If this does not work the next string is removed, and so on until only empty brackets are left. If the search for empty brackets fails, the entire domain literal is removed and rewriting proceeds with the next section of the domain address, if there is one. No asterisks are used in the internal processing of domain literals; when an entire domain literal is replaced by an asterisk the number of asterisks corresponds to the number of elements in the domain literal.

Like normal domain/host specifications, domain literals are also tried in most specific to least specific order. The first rule whose pattern matches will be the one used to rewrite the host/domain specification. If there are two identical patterns in the rules list, the one which appears first will be used.

As an example, suppose the address `dan@[128.6.3.40]` is to be rewritten. The rewriter looks for `[128.6.3.40]`, then `[128.6.3.]`, then `[128.6.]`, then `[128.]`, then `[]`, then `[*.*.*.*]`, and finally the match-all rule “`.`”.

General Database Substitutions (`$(...)`)

A substitution of the form $(*text*) is handled specially. This database is generated with the `crdb` utility. If text is found in the database the corresponding template from the database is substituted. If text does not match an entry in the database the rewrite process fails; it is as if the rewrite rule never matched in the first place. If the substitution is successful the template extracted from the database is re-scanned for additional substitutions. However, additional $(*text*) substitutions from the extracted template are prohibited in order to prevent endless recursive references.

As an example, suppose that the address `jdoe@acme.decnet` matches the following rewrite rule:

```
.DECNET     $($H)
```

Then, the text string acme will be looked up in the general database and the result of the look up, if any, instead used for the rewrite rule's template. Suppose, that the result of looking up acme is `$u%eng.acme.com@tcp-local`. Then the output of the template will be `jdoe@eng.acme.com` (username = `jdoe`, host/domain specification = `eng.acme.com`), and the routing system will be `tcp-local`.

If a general database exists it should be world readable to insure that it operates properly.

Note – This database consists of the files specified with the general database option in the `/imta/tailor` file, which are usually the files `/var/opt/SUNWmail/imta/db/generaldb.*`.

Apply Specified Mapping (`${...}`)

A substitution of the form ${*mapping,argument*} is handled specially. The *mapping,argument* part is used to find and apply a mapping from the IMTA mapping file. The *mapping* field specifies the name of the mapping table to use while *argument* specifies the string to pass to the mapping. The mapping must exist and must set the $Y flag in its output if it is successful; if it doesn't exist or doesn't set $Y the rewrite will fail. If successful the result of the mapping is merged into the template at the current location and re-expanded.

This mechanism allows the IMTA's rewriting process to be extended in various complex ways. For example, the username part of an address can be selectively analyzed and modified, which normally isn't a feature that the IMTA's rewriting process is capable of.

Controlling Error Messages Associated With Rewriting ($?)

The IMTA provides default error messages, when rewriting and channel matching fail. The ability to change these messages can be useful under certain circumstances. For example, if someone tries to send mail to an ethernet router box, it may be considered more informative to say something like "our routers cannot accept mail" rather than the usual "illegal host/domain specified."

A special control sequence can be used to change the error message that is printed if the rule fails. The sequence `$?` is used to specify an error message. Text following the `$?`, up to either an at sign (@), percent sign (%), `$N`, $M, `$Q`, `$C`, `$T`, or `$?` is taken to be the text of the error message to print if the result of this rewrite fails to match any channel. The setting of an error message is “sticky” and lasts through the rewriting process.

A rule that contains a `$?` operates just like any other rule. The special case of a rule containing only a `$?` and nothing else receives special attention: The rewriting process is terminated without changing the mailbox or host portions of the address and the host is looked up as-is in the channel table. This lookup is expected to fail and the error message is returned as a result. For example, if the final rewrite rule in the IMTA configuration file is:

```
$?Unrecognized address; contact postmaster@xyz.com
```

then any unrecognized host or domain specifications that can fail will, in the process of failing, generate the error message: `Unrecognized address; contact postmaster@xyz.com`.

Rewrite Rule Control Sequences

Special control sequences can also appear in rewrite rule templates. These sequences impose additional conditions to the applicability of a given rewrite rule: not only must the pattern portion of the rewrite rule match the host / domain specification being examined, but other aspects of the address being rewritten must meet conditions set by the control sequence or sequences. For instance, the `$E` control sequence requires that the address being rewritten be an envelope address, while the `$F` control sequence requires that it be a forward-pointing address. Thus, the rewrite rule:

```
acme.com     $U@mail.acme.com$E$F
```

only applies to (rewrite) `envelope To:` addresses of the form `user@acme.com`. If a domain / host specification matches the pattern portion of a rewrite rule but doesn’t meet all of the criteria imposed by control sequences in the rule’s template, then the rewrite rule fails and the rewriter continues to look for other applicable rules. This makes possible sets of rewrite rules such as:

```
acme.com     $U@mail.acme.com$Nverify
acme.com     $U%acme.com@verify-daemon
```

which results in messages to `user@acme.com` being passed to the directory channel. However, should the verify channel rewrite a message with the address `user@acme.com`, that message does not again pass through the verify channel. This then allows all mail to `user@acme.com` to pass through the verify channel and for the verify channel to send mail to that address without causing a mail loop.

A summary of control sequences is contained in TABLE 3-5.

TABLE 3-5 Summary of Template Control Sequences

Control Sequence	Effect on Rewrite Rule
$E	Apply only to envelope addresses
$B	Apply only to header and body addresses
$F	Apply only to forward-directed (for example, `To:`) addresses
$R	Apply only to backward-directed (for example, `From:`) addresses
$M *channel*	Apply only if channel is rewriting the address
$N *channel*	Fail if channel *channel* is rewriting the address
$Q *channel*	Apply if sending to channel *channel*
$C *channel*	Fail if sending to channel *channel*
$S	Apply if host is from a source route
$A	Apply if host is to the right of the at sign
$P	Apply if host is to the right of a percent sign
$X	Apply if host is to the left of an exclamation point
$? *errmsg*	If rewriting fails return *errmsg* instead of the default error message

Channel Definitions

The second part of an IMTA configuration file contains the definitions for the channels, themselves. These definitions are collectively referred to as the "channel/host table." Each individual channel definition forms a "channel block," which defines the channels that IMTA can use and the names associated with each channel. Blocks are separated by single blank lines. Comments, but no blank lines, may appear inside a channel block. A channel block contains a list of keywords which define the configuration of a channel. These keywords are referred to as "channel keywords." See TABLE 3-8 for more information.

The following `imta.cnf` file fragment displays a sample channel block:

```
[blank line]
! sample channel block
channelname keyword1 keyword2
routing_system
[blank line]
```

The `routing_system` is an abstract label used to refer to this channel within the rewrite rules.

For detailed information about channel definitions and channel table keywords, refer to the section entitled, “Channel Keywords Table,” and TABLE 3-8 within that section.

Channel Configuration Keywords

The first line of each channel block is composed of the channel name, followed by a list of keywords defining the configuration of the specific channel. The following sections describe keywords and how they control the types of addresses the channel supports. A distinction is made between the addresses used in the transfer layer (the message envelope) and those used in message headers.

Address Interpretation (`bangoverpercent`, `nobangoverpercent`)

Addresses are always interpreted in accordance with RFC 822 and RFC 976. However, there are ambiguities in the treatment of certain composite addresses that are not addressed by these standards. In particular, an address of the form A!B%C can be interpreted as either:

- A as the routing host and C as the final destination host

or

- C as the routing host and A as the final destination host

While RFC 976 implies that it is all right for mailers to interpret addresses using the latter set of conventions, it does not say that such an interpretation is required. In fact, some situations may be better served by the former interpretation.

The `bangoverpercent` keyword forces the former `A!(B%C)` interpretation. The `nobangoverpercent` keyword forces the latter `(A!B)%C` interpretation. `nobangoverpercent` is the default.

Note – This keyword does not affect the treatment of addresses of the form `A!B@C`. These addresses are always treated as `(A!B)@C`. Such treatment is mandated by both RFC 822 and RFC 976.

Routing Information in Addresses (`exproute`, `noexproute`, `improute`, `noimproute`)

The addressing model that IMTA deals with assumes that all systems are aware of the addresses of all other systems and how to get to them. Unfortunately, this ideal is not possible in all cases, such as, when a channel connects to one or more systems that are not known to the rest of the world (for example, internal machines on a private TCP/IP network). Addresses for systems on this channel may not be legal on remote systems outside of the site. If it is desirable to be able to reply to such addresses, they must contain a source route that tells remote systems to route messages through the local machine. The local machine can then (automatically) route the messages to these machines.

The `exproute` keyword (short for “explicit routing“) tells IMTA that the associated channel requires explicit routing when its addresses are passed on to remote systems. If this keyword is specified on a channel, IMTA adds routing information containing the name of the local system (or the current alias for the local system) to all header addresses and all envelope `From:` addresses that match the channel. `noexproute`, the default, specifies that no routing information should be added.

The `EXPROUTE_FORWARD` option can be used to restrict the action of `exproute` to backward-pointing addresses, if desired. Another scenario occurs when IMTA connects to a system via a channel that cannot perform proper routing for itself. In this case, all addresses associated with other channels need to have routing inserted into them when they are used in mail sent to the channel that connects to the incapable system.

Implicit routing and the `improute` keyword is used to handle this situation. IMTA knows that all addresses matching other channels need routing when they are used in mail sent to a channel marked `improute`. `noimproute`, the default, specifies that no routing information should be added to addresses in messages going out on the specified channel. The `IMPROUTE_FORWARD` option can be used to restrict the action of `improute` to backward-pointing addresses, if desired.

The `exproute` and `improute` keywords should be used sparingly. It makes addresses longer, more complex, and may defeat intelligent routing schemes used by other systems. Explicit and implicit routing should not be confused with specified routes. Specified routes are used to insert routing information from rewrite rules into addresses. This is activated by the special `A@B@C` rewrite rule template.

Specified routes, when activated, apply to all addresses, both in the header and the envelope. Specified routes are activated by particular rewrite rules and as such are usually independent of the channel currently in use. Explicit and implicit routing, on the other hand, are controlled on a per-channel basis and the route address inserted is always the local system.

Address Rewriting Upon Message Dequeue (`connectalias`, `connectcanonical`)

IMTA normally rewrites addresses as it enqueues messages to its channel queues. No additional rewriting is done during message dequeue. This presents a potential problem when host names change while there are messages in the channel queues still addressed to the old name.

- The `connectalias` keyword tells IMTA to deliver to whatever host is listed in the recipient address. This is the default. `connectcanonical` forces IMTA to run the address through the rewrite rules one additional time and use the resulting host.

Channel Directionality (`master`, `slave`, `bidirectional`)

Three keywords are used to specify whether a channel is served by a master program (`master`), a slave program (`slave`), or both (`bidirectional`). The default, if none of these keywords are specified, is `bidirectional`. These keywords determine whether IMTA initiates delivery activity when a message is queued to the channel.

The use of these keywords reflect certain fundamental characteristics of the corresponding channel program or programs. The descriptions of the various channels IMTA supports indicate when and where these keywords should be used.

Channel Service Periodicity (`immediate`, `immnonurgent`, `immnormal`, `immurgent`, `periodic`, `period`)

If a channel is capable of master-mode operations (as specified with the `master` keyword), such operations may be initiated either by a periodic service job or on demand as delivery is needed:

- `immediate`, which is the default, specifies that jobs should run on demand for messages of appropriate urgency.
- `periodic` inhibits initiation of delivery jobs on demand for the channel it is associated with, regardless of priority.

  What *appropriate urgency* means is controlled by the keywords:

- `immurgent` enables immediate delivery processing on messages with a priority setting of urgent. Messages with a lower priority must wait for periodic processing.
- `immnormal` enables immediate delivery for messages with normal or urgent priority (`immnormal` is the default keyword with `immediate`).
- `immnonurgent` enables immediate delivery for urgent, normal, and nonurgent messages.

Thus the default behavior (`immediate immnormal`) enables immediate processing for all but nonurgent or lower priority messages.

Delivery via periodic service jobs is always possible unless the channel is marked with the `slave` keyword. Channels capable of master-mode operation are periodically checked for pending messages by periodic service jobs. These jobs run at fixed intervals, usually every four hours, though you can change this interval. On UNIX systems, the interval is determined in the `crontab` entry for the post job.

Not all channels need service at the same intervals. For example, a channel might see little traffic and be expensive to service. Servicing such a channel at longer intervals than that of a single period between periodic jobs can lower the cost of operation without significantly affecting the quality of service.

In another case, one particular channel may see very heavy traffic and may require frequent service, while other channels need servicing much less often. In this situation it may be appropriate to service the heavily used channel more often than any other.

The `period` keyword can be used to control how often a channel is serviced. This keyword must be followed by an integer value *N*. The channel is then serviced by every *Nth* service job. The default value of the period keyword is `1`, which means that every periodic service job checks the channel for pending messages.

Message Size Affecting Priority (`urgentblocklimit`, `normalblocklimit`, `nonurgentblocklimit`)

The `urgentblocklimit`, `normalblocklimit`, and `nonurgentblocklimit` keywords may be used to downgrade the priority of messages based on size. This priority, in turn, may affect whether the message is processed immediately, or whether it is left to wait for processing until the next periodic job runs.

The `urgentblocklimit` keyword instructs IMTA to downgrade messages larger than the specified size to `normal` priority. The `normalblocklimit` keyword instructs IMTA to downgrade messages larger than the specified size to `nonurgent` priority. The `nonurgentblocklimit` keyword instructs IMTA to downgrade messages larger than the specified size to lower than `nonurgent` priority (second class priority), meaning that the messages always wait for the next periodic job for further processing.

Channel Connection Information Caching (`cacheeverything`, `cachesuccesses`, `cachefailures`, `nocache`)

SMTP channels maintain a cache containing a history of prior connection attempts. This cache is used to avoid reconnecting multiple times to inaccessible hosts, which can waste lots of time and delay other messages. The cache normally records both connection successes and failures. (Successful connection attempts are recorded in order to offset subsequent failures; for example, a host that succeeded before but fails now doesn't warrant as long a delay before making another connection attempt as does one that has never been tried or one that has failed previously.)

However, this caching strategy is not necessarily appropriate for all situations. For example, a SMTP router channel that is used to connect to a single unpredictable host does not benefit from caching. Therefore channel keywords are provided to adjust IMTA's cache.

The `cacheeverything` keyword enables all forms of caching and is the default. `nocache` disables all caching. `cachefailures` enables caching of connection failures but not successes. Finally, `cachesuccesses` caches only successful connections. This last keyword is equivalent to `nocache` for channels.

Number of Addresses or Message Files to Handle per Service Job or File (`addrsperjob`, `filesperjob`, `maxjobs`)

When a message is enqueued to a channel the job controller normally starts one master process per each channel. If the channel is processed on a periodic basis, one master process per channel is started.

A single master process might not be sufficient to ensure prompt delivery of all messages, however. In particular, fax messages may take a long time to deliver; if multiple fax modems are available, it is not efficient to use a single process and a single modem.

The `addrsperjob` and `filesperjob` keywords can be used to create additional master processes. Each of these keywords take a single positive integer parameter which specifies how many addresses or queue entries (files) must be sent to the associated channel before more than one master process is created to handle them. If a value less than or equal to zero is given, it is interpreted as a request to queue only one service job. Not specifying a keyword defaults to a value of 0. The effect of these keywords is maximized; the larger number computed is the number of service jobs that are actually created.

The `addrsperjob` keyword computes the number of service jobs to start by dividing the total number of `To:` addressees in all entries by the given value. The `filesperjob` keyword divides the number of actual queue entries or files by the given value. The number of queue entries resulting from a given message is controlled by a large number of factors, including but not limited to the use of the `single` and `single_sys` keywords and the specification of header-modifying actions in mailing lists.

The `maxjobs` keyword places an upper-limit on the total number of service jobs that can be created. This keyword must be followed by an integer value; if the computed number of service jobs is greater than this value, only `maxjobs` processes are actually created. If `maxjobs` is not specified, the default for this value is `100`. Normally `maxjobs` is set to a value that is less than or equal to the total number of jobs that can run simultaneously in whatever service queue or queues the channel uses.

For example, if a message with four recipient addresses is queued to a channel marked `addrsperjob 2` and `maxjobs 5`, a total of two service jobs are created. But if a message with 23 recipient addresses is queued to the same channel, only five jobs are created because of the `maxjobs` restriction.

Note – These keywords affect the creation of both periodic and immediate service jobs. In the case of periodic jobs, the number of jobs created is calculated from the total number of messages in the channel queue. In the case of immediate service jobs, the calculation is based only on the message being entered into the queue at the time.

The `addrsperjob` keyword is generally useful only on channels that provide per-address service granularity. Currently this is limited to fax channels.

Multiple Addresses (`multiple`, `addrsperfile`, `single`, `single_sys`)

The IMTA allows multiple destination addresses to appear in each queued message. Some channel programs, however, may only be able to process messages with one recipient, or with a limited number of recipients, or with a single destination system per message copy. For example, the SMTP channels master program only establishes a connection to a single remote host in a given transaction, so only addresses to that host can be processed (this despite the fact that a single channel is typically used for all SMTP traffic).

Another example is that some SMTP servers may impose a limit on the number of recipients they can handle at one time, and they may not be able to handle this type of error.

The keywords `multiple`, `addrsperfile`, `single`, and `single_sys` can be used to control how multiple addresses are handled. `single` means that a separate copy of the message should be created for each destination address on the channel. `single_sys` creates a single copy of the message for each destination system used. `multiple`, the default, creates a single copy of the message for the entire channel.

Note – At least one copy of each message is created for each channel the message is queued to, regardless of the keywords used.

The `addrsperfile` keyword is used to put a limit on the maximum number of recipients that can be associated with a single message file in a channel queue, thus limiting the number of recipients that are processed in a single operation. This keyword requires a single-integer argument specifying the maximum number of recipient addresses allowed in a message file; if this number is reached, IMTA automatically creates additional message files to accommodate them. (The default multiple keyword corresponds to imposing no limit on the number of recipients in a message file.)

Expansion of Multiple Addresses (`expandlimit`)

Most channels support the specification of multiple recipient addresses in the transfer of each inbound message. The specification of many recipient addresses in a single message may result in delays in message transfer processing (“online” delays). If the delays are long enough, network timeouts can occur, which in turn can lead to repeated message submission attempts and other problems.

IMTA provides a special facility to force deferred (“offline”) processing if more than a given number of addresses are specified for a single message. Deferral of message processing can decrease online delays enormously. Note, however, that the processing overhead is only deferred, not avoided completely.

This special facility is activated by using a combination of the generic reprocessing channel and the `expandlimit` keyword. The `expandlimit` keyword takes an integer argument that specifies how many addresses should be accepted in messages coming from the channel before deferring processing. The default value is infinite if the `expandlimit` keyword is not specified. A value of 0 forces deferred processing on all incoming addresses from the channel.

The `expandlimit` keyword must not be specified on the local channel or the reprocessing channel itself; the results of such a specification are unpredictable. The reprocessing channel is used to perform the deferred processing and must be added to the configuration file in order for the `expandlimit` keyword to have any effect. If your configuration was built by the IMTA configuration utility, then you should already have such a channel.

Multiple Subdirectories (`subdirs`)

By default, all messages queued to a channel are stored as files in the directory `/imta/queue/channel-name`, where `channel-name` is the name of the channel. However, a channel that handles a large number of messages and tends to build up a large store of message files waiting for processing, for example, a TCP/IP channel, may get better performance out of the file system if those message files are spread across a number of subdirectories. The `subdirs` channel keyword provides this capability: it should be followed by an integer that specifies the number of subdirectories across which to spread messages for the channel, for example, `tcp_local single_sys smtp subdirs 10`.

Service Job Queue (`queue`)

IMTA creates service jobs (channel master programs) to deliver messages. The job controller, which launches these jobs, associates them with queues. Queue types are defined in the job_controller.cnf file. The queue type with which each channel's master program is associated can be selected on a channel-by-channel basis, using the `queue` keyword. The `queue` keyword must be followed by the name of the queue type to which delivery jobs for the current channel should be queued. The name of the queue type should *not* contain more than 12 characters. If the `queue` keyword is omitted, then the queue used is the default queue, the first queue listed in the job controller configuration file.

Deferred Delivery Dates (`deferred`, `nodeferred`)

The `deferred` channel keyword implements recognition and honoring of the `Deferred-delivery:` header. Messages with a `deferred` delivery date in the future are held in the channel queue until they either expire and are returned or the deferred delivery date is reached. See RFC 1327 for details on the format and operation of the `Deferred-delivery:` header.

nodeferred is the default. It is important to realize that while support for deferred message processing is mandated by RFC 1327, actual implementing of it effectively lets people use the mail system as an extension of their disk quota.

Undeliverable Message Notification Times (notices)

The notices keyword controls the amount of time an undeliverable message is silently retained in a given channel queue. IMTA is capable of returning a series of warning messages to the originator and, if the message remains undeliverable, IMTA eventually returns the entire message.

The keyword is followed by a list of up to five monotonically increasing integer values. These values refer to the message ages at which warning messages are sent. The ages have units of days if the RETURN_UNITS option is 0 or not specified in the option file; or hours if the RETURN_UNITS option is 1. When an undeliverable message attains or exceeds the last listed age, it is returned (bounced).

When a message attains any of the other ages, a warning notice is sent. The default if no notices keyword is given is to use the notices setting for the local channel. If no setting has been made for the local channel, then the defaults 3, 6, 9, 12 are used, meaning that warning messages are sent when the message attains the ages 3, 6, and 9 days (or hours) and the message is returned after remaining in the channel queue for more than 12 days (or hours).

Note – The syntax for the notices keyword uses no punctuation. For example, the default return policy is expressed as: notices 3 6 9 12.

The following line specifies that if messages are enqueued to the tcp_local channel and deferred for later reprocessing, transient failure delivery status notifications will be generated after 1 and 2 days. If the message is still not delivered after 5 days, it will be returned to its originator.

```
tcp_local charset7 us-ascii charset8 iso-8853-1 notices 1 2 3 mail.alpha.com
```

The defaults channel appears immediately after the first blank line in the configuration file, usually /imta/table/imta.cnf. It is important that a blank line appear before and after the line defaults notices....

Returned Messages (`sendpost`, `nosendpost`, `copysendpost`, `errsendpost`)

A channel program may be unable to deliver a message because of long-term service failures or invalid addresses. When this failure occurs, the IMTA channel program returns the message to the sender with an accompanying explanation of why the message was not delivered. Optionally, a copy of all failed messages is sent to the local postmaster. This is useful for monitoring message failures, but it can result in lots of traffic for the postmaster to deal with.

The keywords `sendpost`, `copysendpost`, `errsendpost`, and `nosendpost` control the sending of failed messages to the postmaster. `sendpost` tells IMTA to send a copy of all failed messages to the postmaster unconditionally. `copysendpost` instructs IMTA to send a copy of the failure notice to the postmaster unless the originator address on the failing message is blank, in which case, the postmaster gets copies of all failed messages except those messages that are actually themselves bounces or notifications.

`errsendpost` instructs IMTA to send a copy of the failure notice only to the postmaster, when the notice cannot be returned to the originator. No failed messages are ever sent to the postmaster if `nosendpost` is specified. The default, if none of these keywords is specified, is to send a copy of failed mail messages to the postmaster, unless error returns are completely suppressed with a blank `Errors-to:` header or a blank envelope `From:` address.This default behavior does not correspond to any of the keyword settings.

Warning Messages (`warnpost`, `nowarnpost`, `copywarnpost`, `errwarnpost`)

In addition to returning messages, IMTA sometimes sends warnings detailing messages that it has been unable to deliver. This is generally due to timeouts based on the setting of the notices channel keyword, although in some cases channel programs may produce warning messages after failed delivery attempts. The warning messages contain a description of what's wrong and how long delivery attempts will continue. In most cases they also contain the headers and the first few lines of the message in question.

Optionally, a copy of all warning messages is sent to the local postmaster. This can be somewhat useful for monitoring the state of the various queues, although it does result in lots of traffic for the postmaster to deal with. The keywords `warnpost`, `copywarnpost`, `errwarnpost`, and `nowarnpost` are used to control the sending of warning messages to the postmaster.

- `warnpost`–Tells IMTA to send a copy of all warning messages to the postmaster unconditionally

- `copywarnpost`–Instructs IMTA to send a copy of the warning to the postmaster, unless the originator address on the undelivered message is blank.

  In this case, the postmaster gets copies of all warnings of undelivered messages except for undelivered messages that are actually themselves bounces or notifications.
- `errwarnpost`–Instructs IMTA to send only a copy of the warning to the postmaster when the notice cannot be returned to the originator.

No warning messages are ever sent to the postmaster if `nowarnpost` is specified. The default, if none of these keywords is specified, is to send a copy of warnings to the postmaster unless warnings are completely suppressed with a blank `Warnings-to:` header or a blank envelope `From:` address. This default behavior does not correspond to any of the keyword settings.

Postmaster Returned Message Content (`postheadonly`, `postheadbody`)

When a channel program or the periodic message return job returns messages to both the postmaster and the original sender, the postmaster copy can either be the entire message or just the headers. Restricting the postmaster copy to just the headers adds an additional level of privacy to user mail. However, this by itself does not guarantee message security; postmasters and system managers are typically in a position where the contents of messages can be read using `root` system privileges, if they so choose.

The keywords `postheadonly` and `postheadbody` are used to control what gets sent to the postmaster. `postheadbody` returns both the headers and the contents of the message. It is the default. `postheadonly` causes only the headers to be sent to the postmaster.

Triggering New Threads in Multithreaded Channels (`threaddepth`)

The multithreaded SMTP client sorts outgoing messages to different destinations to different threads. The `threaddepth` keyword may be used to instruct IMTA's multithreaded SMTP client to handle only the specified number of messages in any one thread, using additional threads even for messages all to the same destination (hence normally all handled in one thread).

Channel Protocol Selection (`smtp`, `nosmtp`)

These options specify whether or not a channel supports the SMTP protocol and what type of SMTP line terminator IMTA expects to see as part of that protocol. `nosmtp` means that the channel doesn't support SMTP; all the rest of these keywords imply SMTP support.

The selection of whether or not to use the SMTP protocol is implicit for most channels; the correct protocol is chosen by the use of the appropriate channel program or programs. Some gateway systems use the Simple Mail Transfer Protocol (SMTP) described in RFC 821 as a message envelope, while others may not use an envelope format. The result is that all envelope information is derived from the RFC 822 message header, which is present in all cases. The `smtp` keyword is used to tell the channel master programs to put a batch SMTP header on the message. The `nosmtp` keyword inhibits the generation of the batch SMTP header. `nosmtp` is the default.

The keyword `smtp` is mandatory for all SMTP channels. The keywords `smtp_cr`, `smtp_crlf`, and `smtp_lf` may be used on SMTP channels to specify what character sequences to accept as line terminators. `smtp_crlf` means that lines must be terminated with a carriage return (CR) line feed (LF) sequence. `smtp_lf` or `smtp` means that an LF without a preceding CR is accepted. Finally, `smtp_cr` means that a CR is accepted without a following LF. It is normal to use CRLF sequences as the SMTP line terminator, and this is what IMTA always generates; this option affects only the handling of incoming material.

SMTP EHLO command (`ehlo`, `checkehlo`, `noehlo`)

The SMTP protocol has recently been extended (RFC 1651) to allow for negotiation of additional commands. This is done via the new `EHLO` command, which replaces RFC 821's `HELO` command. Extended SMTP servers respond to `EHLO` by providing a list of the extensions they support. Unextended servers return an unknown command error and the client then sends the old `HELO` command instead.

This fallback strategy normally works well with both extended and unextended servers. Problems can arise, however, with servers that do not implement SMTP according to RFC 821. In particular, some incompliant servers are known to drop the connection on receipt of an unknown command.

The SMTP client implements a strategy whereby it attempts to reconnect and use `HELO` when any server drops the connection on receipt of an `EHLO`. However, this strategy may not work if the remote server not only drops the connection but also goes into a problematic state upon receipt of `EHLO`.

The channel keywords `ehlo`, `noehlo`, and `checkehlo` are provided to deal with such situations. `EHLO` tells IMTA to use the `ehlo` command on all initial connection attempts. `noehlo` disables all use of the `EHLO` command. `checkehlo` tests the response banner returned by the remote SMTP server for the string "ESMTP." If this string is found, `EHLO` is used; if not, `HELO` is used. The default behavior is to use `EHLO` on all initial connection attempts, unless the banner line contains the string "fire away," in which case `HELO` is used.

Note – There is no keyword corresponding to this default behavior, which lies between the behaviors resulting from the `ehlo` and `checkehlo` keywords.

SMTP `VRFY` Commands (`domainvrfy`, `localvrfy`, `novrfy`)

These keywords control IMTA's use of the `VRFY` command in its SMTP client. Under normal circumstances there is no reason to issue a `VRFY` command as part of an SMTP dialogue. The SMTP `MAIL TO` command should perform the same function that `VRFY` does and return an appropriate error. However, servers exist that can accept any address in a `MAIL TO` (and bounce it later), whereas these same servers perform more extensive checking as part of a `VRFY` command.

The IMTA can be configured to issue `SMTP VRFY` commands. The keyword `domainvrfy` causes a `VRFY` command to be issued with a full address (`user@host`) as its argument. The `localvrfy` keyword causes IMTA to issue a `VRFY` command with just the local part of the address (`user. novrfy` is the default).

TCP/IP Port Number (`port`)

SMTP over TCP/IP channels normally connect to port 25 when sending messages. The `port` keyword may be used to instruct an SMTP over TCP/IP channel to connect to a non-standard port.

TCP/IP MX Record Support (`mx`, `nomx`, `defaultmx`, `randommx`, `nonrandommx`)

Some TCP/IP networks support the use of `MX` (mail forwarding) records and some do not. Some TCP/IP channel programs can be configured not to use `MX` records if they are not provided by the network that the IMTA system is connected to. `randommx` specifies that `MX` lookups should be done and `MX` record values of equal precedence should be processed in random order. nonrandommx specifies that MX lookups should be done and MX values of equal precedence should be processed in the same order in which they were received.

The mx keyword is currently equivalent to nonrandommx; it may change to be equivalent to randommx in a future release. The nomx keyword disables MX lookups. The defaultmx keyword specifies that mx should be used if the network says that MX records are supported. defaultmx is the default on channels that support MX lookups in any form.

Specify a Last Resort Host (lastresort)

The lastresort keyword is used to specify a host to which to connect when all other connection attempts fail. In effect this acts as an MX record of last resort. This is only useful on SMTP channels.

Reverse DNS and IDENT Lookups on Incoming SMTP Connections (identtcp, identtcpnumeric, identnone, identnonenumeric)

The identtcp keyword tells IMTA to perform a connection and lookup using the IDENT protocol (RFC 1413). The information obtained from the IDENT protocol (usually the identity of the user making the SMTP connection) is then inserted into the Received: headers of the message, with the hostname corresponding to the incoming IP number, as reported from a DNS reverse lookup.

The identtcpnumeric keyword tells IMTA to perform a connection and lookup using the IDENT protocol (RFC 1413). The information obtained from the IDENT protocol (usually the identity of the user making the SMTP connection) is then inserted into the Received: headers of the message, with the actual incoming IP number. No DNS reverse lookup on the IP number is performed.

Note – The remote system must be running an IDENT server in order for the IDENT lookup caused by either identtcp or identtcpnumeric to be useful.

When comparing identtcp with identtcpnumeric, the DNS reverse lookup called for with identtcp incurs some additional overhead to obtain the more “user-friendly” hostname. The actual IP number may also be considered more definitive than the hostname returned from the DNS reverse lookup.

The `identnone` keyword disables this `IDENT` lookup, but does do IP-to-hostname translation. The `identnonenumeric` keyword disables this `IDENT` lookup and inhibits the usual DNS reverse lookup translation of IP number to hostname, and may therefore result in a performance improvement at the cost of less user-friendly information in the `Received:` headers. `identnone` is the default. These keywords are only useful on SMTP channels that run over TCP/IP.

Select an Alternate Channel for Incoming Mail (`switchchannel`, `allowswitchchannel`, `noswitchchannel`)

When an IMTA server accepts an incoming connection from a remote system it must choose a channel with which to associate the connection. Normally this decision is based on the transfer used; for example, an incoming TCP/IP connection is automatically associated with the `tcp_local` channel.

This convention breaks down, however, when multiple outgoing channels with different characteristics are used to handle different systems over the same transfer. When this happens, incoming connections are not associated with the same channel as outgoing connections, and the result is that the corresponding channel characteristics are not associated with the remote system.

The `switchchannel` keyword provides a way to eliminate this difficulty. If `switchchannel` is specified on the server's initial channel (`tcp_local`), the name of the originating host is matched against the channel table; if it matches, the source channel changes accordingly. The source channel may change to any channel marked `switchchannel` or `allowswitchchannel` (the default). `noswitchchannel` specifies that no channel switching should be done to or from the channel.

Specification of `switchchannel` on anything other than a channel that a server associates with by default has no effect. At present, `switchchannel` only affects SMTP channels, but there are actually no other channels where `switchchannel` would be reasonable.

Note – When the `switchchannel` is specified, the name of the originating host is obtained by a DNS reverse lookup translation of the IP address to hostname. Consequently, this keyword is useful for setting up anti-spamming but it may affect performance.

Hostname to Use When Correcting Incomplete Addresses (`remotehost`, `noremotehost`)

The IMTA often receives from misconfigured or incompliant mailers and SMTP clients addresses that do not contain a domain name. IMTA attempts to make such addresses legal before allowing them to pass further. IMTA does this by appending a domain name to the address (for example, appends `@acme.com` to `mrochek`). In the case of the SMTP server, however, the two logical choices for the domain name are:

- Local host name
- Remote host name reported by the client SMTP

Either of these two choices is likely to be correct, as both may occur operationally with some frequency. The use of the remote host's domain name is appropriate when dealing with improperly configured SMTP clients. The use of the local host's domain name is appropriate when dealing with a lightweight remote mail client such as a POP or IMAP client that uses SMTP to post messages.

The best that IMTA can do is to allow the choice to be made on a channel-by-channel basis. The `remotehost` channel keyword specifies that the remote host's name should be used. The `noremotehost` channel keyword specifies that the local host's name should be used. `noremotehost` is the default.

The `switchchannel` keyword as described, in the preceding section, "Select an Alternate Channel for Incoming Mail (switchchannel, allowswitchchannel, noswitchchannel)" can be used to associate incoming SMTP connections with a particular channel. This facility can be used to group remote mail clients on a channel where they can receive proper treatment. Alternatively, it is simpler to deploy standards-compliant remote mail clients (even if a multitude of incompliant clients are in use) rather than attempting to fix the network-wide problem on your IMTA hosts.

Eight-Bit Capability (`eightbit`, `eightnegotiate`, `eightstrict`, `sevenbit`)

Some transfers restrict the use of characters with ordinal values greater than 127 (decimal). Most notably, some SMTP servers strip the high bit and thus garble messages that use characters in this eight-bit range. IMTA provides facilities to automatically encode such messages so that troublesome eight-bit characters do not appear directly in the message. This encoding can be applied to all messages on a given channel by specifying the `sevenbit` keyword. A channel should be marked `eightbit` if no such restriction exists.

Some transfers, such as extended SMTP, may actually support a form of negotiation to determine if eight-bit characters can be transmitted. The `eightnegotiate` keyword can be used to instruct the channel to encode messages when negotiation fails. This is the default for all channels; channels that do not support negotiation assume that the transfer is capable of handling eight-bit data. The `eightstrict` keyword tells IMTA to reject any messages that contain unnegotiated eight-bit data.

Automatic Character Set Labeling (`charset7`, `charset8`)

The `MIME` specification provides a mechanism to label the character set used in a plain text message. Specifically, a `charset=` parameter can be specified as part of the `Content-type:` header line. Various character set names are defined in `MIME`, including US-ASCII (the default), ISO-8859-1, ISO-8859-2, and so on.

Some existing systems and user agents do not provide a mechanism for generating these character set labels; as a result, some plain text messages may not be properly labeled. The `charset7` and `charset8` channel keywords provide a per-channel mechanism to specify character set names to be inserted into message headers. Each keyword requires a single argument giving the character set name. The names are not checked for validity.

Note – Character set conversion can only be done on character sets specified in the character set definition file `charsets.txt` found in the IMTA table directory, `/imta/table/charsets.txt`. The names defined in this file should be used if possible.

The `charset7` character set name is used if the message contains only seven-bit characters; `charset8` is used if eight-bit data is found in the message. If the appropriate keyword is not specified, no character set name is inserted into the `Content-type:` header lines.

These character set specifications never override existing labels; that is, they have no effect if a message already has a character set label or is of a type other than text. It is usually appropriate to label IMTA local channels as follows:

```
l ... charset7 US-ASCII charset8 ISO-8859-1 ...
hostname
```

If there is no Content-type header in the message, it is added. This keyword also adds the MIME-version: header if it is missing.

Message Line Length Restrictions (`linelength`)

The SMTP specification allows for lines of text containing up to 1000 bytes. However, some transfers may impose more severe restrictions on line length. The `linelength` keyword provides a mechanism for limiting the maximum permissible message line length on a channel-by-channel basis. Messages queued to a given channel with lines longer than the limit specified for that channel are automatically encoded.

The various encodings available in the IMTA always result in a reduction of line length to fewer than 80 characters. The original message may be recovered after such encoding is done by applying an appropriating decoding filter.

Note – Encoding can only reduce line lengths to fewer than 80 characters. For this reason, specification of line length values less than 80 may not actually produce lines with lengths that comply with the stated restriction.

Channel-specific Use of the Reverse Database (`reverse`, `noreverse`)

The `reverse` keyword tells IMTA that addresses in messages queued to the channel should be checked against, and possibly modified, by the address reversal database or REVERSE mapping, if either exists. `noreverse` exempts addresses in messages queued to the channel from address reversal processing. The `reverse` keyword is the default.

Inner Header Rewriting (`noinner`, `inner`)

The contents of header lines are only interpreted when necessary. However, `MIME` messages can contain multiple sets of message headers as a result of the ability to imbed messages within messages (message/RFC822). IMTA normally only interprets and rewrites the outermost set of message headers. IMTA can optionally be told to apply header rewriting to inner headers within the message as well.

This behavior is controlled by the use of the `noinner` and `inner` keywords. `noinner` tells IMTA not to rewrite inner message header lines. It is the default. `inner` tells IMTA to parse messages and rewrite inner headers. These keywords can be applied to any channel.

Restricted Mailbox Encoding (`restricted`, `unrestricted`)

Some mail systems have difficulty dealing with the full spectrum of addresses allowed by RFC 822. A particularly common example of this is sendmail-based mailers with incorrect configuration files. Quoted local-parts (or mailbox specifications) are a frequent source of trouble:

```
"smith, ned"@xyz.com
```

This is such a major source of difficulty that a methodology was laid out in RFC 1137 to work around the problem. The basic approach is to remove quoting from the address, then apply a translation that maps the characters requiring quoting into characters allowed in an atom (see RFC 822 for a definition of an atom as it is used here). For example, the preceding address would become:

```
smith#m#_ned@xyz.com
```

The `restricted` channel keyword tells IMTA that the channel connects to mail systems that require this encoding. IMTA then encodes quoted local-parts in both header and envelope addresses as messages are written to the channel. Incoming addresses on the channel are decoded automatically. The `unrestricted` keyword tells IMTA not to perform RFC 1137 encoding and decoding. `unrestricted` is the default.

Note – The `restricted` keyword should be applied to the channel that connects to systems unable to accept quoted local-parts. It should not be applied to the channels that actually generate the quoted local-parts. (It is assumed that a channel capable of generating such an address is also capable of handling such an address.)

Trimming Message Header Lines (`headertrim`, `noheadertrim`, `headerread`, `noheaderread`, `innertrim`, `noinnertrim`)

The IMTA provides per-channel facilities for trimming or removing selected message header lines from messages. This is done through a combination of a channel keyword and an associated header option file or two. The `headertrim` keyword instructs the IMTA to consult a header option file associated with the channel and to trim the headers on messages queued to the channel accordingly, after the messages are processed. The `noheadertrim` keyword bypasses header trimming. `noheadertrim` is the default.

The `innertrim` keyword instructs the IMTA to perform header trimming on inner message parts, for example, embedded MESSAGE/RFC822 parts. The `noinnertrim` keyword, which is the default, tells the IMTA not to perform any header trimming on inner message parts.

The `headerread` keyword instructs the IMTA to consult a header option file associated with the channel and to trim the headers on messages queued to the channel accordingly, before the messages are processed. Note that `headertrim` header trimming, on the other hand, is applied after the messages have been processed. The `noheaderread` keyword bypasses message enqueue header trimming. `noheaderread` is the default.

Caution – Stripping away vital header information from messages may cause improper operation of the IMTA. Be extremely careful when selecting headers to remove or limit. This facility exists because there are occasional situations where selected header lines must be removed or otherwise limited. Before trimming or removing any header line, be sure that you understand the usage of that header line and have considered the possible implications of its removal.

Header options files for the `headertrim` and `innertrim` keywords have names of the form `channel_headers.opt` with *channel*, the name of the channel with which the header option file is associated. Similarly, header options files for the `headerread` keyword have names of the form `channel_read_headers.opt`. These files are stored in the IMTA configuration directory, `/etc/opt/SUNWmail/imta/`.

Encoding Header (`ignoreencoding`, `interpretencoding`)

IMTA can convert various nonstandard message formats to `MIME` via the `Yes CHARSET-CONVERSION`. In particular, the RFC 1154 format uses a nonstandard `Encoding:` header. However, some gateways emit incorrect information on this header line, with the result that sometimes it is desirable to ignore this header. The `ignoreencoding` keyword instructs the IMTA to ignore any `Encoding:` header.

Note – Unless the IMTA has a `CHARSET-CONVERSION` enabled, such headers are ignored in any case. The `interpretencoding` keyword instructs the IMTA to pay attention to any `Encoding:` header, if otherwise configured to do so, and is the default.

Generation of `X-Envelope-to:` Header Lines (`x_env_to`, `nox_env_to`)

The `x_env_to` and `nox_env_to` keywords control the generation or suppression of `X-Envelope-to:` header lines on copies of messages queued to a specific channel. The `x_env_to` keyword enables generation of these headers while the `nox_env_to` will remove such headers from enqueued messages. The default is `nox_env_to`.

Envelope to Address in `Received:` Header (`receivedfor`, `noreceivedfor`)

The `receivedfor` keyword instructs the IMTA that if a message is addressed to just one envelope recipient, to include that envelope to address in the `Received:` header it constructs. `receivedfor` is the default. The `noreceivedfor` keyword instructs IMTA to construct `Received:` headers without including any envelope addressee information.

Blank Envelope Return Addresses (`returnenvelope`)

The `returnenvelope` keyword takes a single integer value, which is interpreted as a set of bit flags. Bit 0 (value = 1) controls whether or not return notifications generated by IMTA are written with a blank envelope address or with the address of the local postmaster. Setting the bit forces the use of the local postmaster address, clearing the bit forces the use of a blank address.

Note – The use of a blank address is mandated by RFC 1123. However, some systems do not properly handle blank envelopes `From:` address and may require the use of this option.

Bit 1 (value = 2) controls whether or not IMTA replaces all blank envelope addresses with the address of the local postmaster. This is used to accommodate incompliant systems that don't conform to RFC 821, RFC 822, or RFC 1123.

Mapping `Reply-to:` Header (`usereplyto`)

The `usereplyto` keyword controls the mapping of the `Reply-to:` header. The default is `usereplyto 0`, which means to use the channel default behavior (which varies from channel to channel). TABLE 3-6 indicates the mapping specifications for the `Reply-to:` header:

TABLE 3-6 Reply-to: Header Mapping Options

Value	Action
-1	Never map `Reply-to:` addresses to anything.
0	Use the channel default mapping of `Reply-to:` addresses; (varies from channel to channel). This is the default.
1	Map `Reply-to:` to `From:` if no usable `From:` address exists.
2	If there is a usable `Reply-to:` address, then map it to `From:`; otherwise, fall back to the `From:` address.

Mapping `Resent-` Headers When Gatewaying to non-RFC 822 Environments (`useresent`)

The `useresent` keyword controls the use of `Resent-` headers when gatewaying to environments that do not support RFC 822 headers. This keyword takes a single integer-valued argument. TABLE 3-7 lists the values used for mapping the `Resent-` headers:

TABLE 3-7 `Resent-` Headers Mapping Options

Value	Action
+2	Use any `Resent-` headers that are present to generate address information.
+1	Use only `Resent-From` headers to generate address information; all other `Resent-` headers are ignored.
0	Do not use `Resent-` headers to generate address information. This is the default.

Comments in Address Message Headers (`commentinc`, `commentomit`, `commentstrip`, `commenttotal`)

IMTA interprets the contents of header lines only when necessary. However, all registered headers containing addresses must be parsed in order to rewrite and eliminate shortform addresses and otherwise convert them to legal addresses. During this process, comments (strings enclosed in parentheses) are extracted and may optionally be modified or excluded when the header line is rebuilt.

This behavior is controlled by the use of the `commentinc`, `commentomit`, `commentstrip`, and `commenttotal` keywords. `commentinc` tells IMTA to retain comments in header lines. It is the default. `commentomit` tells IMTA to remove any comments from addressing headers, for example, `To:`, `From:`, or `Cc:` headers.

`commenttotal` tells IMTA to remove any comments from all headers, including `Received:` headers; as such, this keyword is not normally useful or recommended. `commentstrip` tells IMTA to strip any non-atomic characters from all comment fields. These keywords can be applied to any channel.

Personal Names in Address Message Headers (`personalinc`, `personalomit`, `personalstrip`)

During the rewriting process, all registered headers containing addresses must be parsed in order to rewrite and eliminate shortform addresses and otherwise convert them to legal addresses. During this process personal names (strings preceding angle-bracket-delimited addresses) are extracted and can be optionally modified or excluded when the header line is rebuilt.

This behavior is controlled by the use of the `personalinc`, `personalomit`, and `personalstrip` keywords. `personalinc` tells IMTA to retain personal names in the headers. It is the default. `personalomit` tells IMTA to remove all personal names. `personalstrip` tells IMTA to strip any nonatomic characters from all personal name fields. These keywords can be applied to any channel.

Two- or Four-digit Date Conversion (`datefour`, `datetwo`)

The original RFC 822 specification called for two-digit years in the date fields in message headers. This was later changed to four digits by RFC 1123. However, some older mail systems cannot accommodate four-digit dates. In addition, some newer mail systems can no longer tolerate two-digit dates.

Note – Systems that cannot handle both formats are in violation of the standards.

The `datefour` and `datetwo` keywords control IMTA's processing of the year field in message header dates. `datefour`, the default, instructs IMTA to expand all year fields to four digits. Two digit dates with a value less than 50 have 2000 added, while values greater than 50 have 1900 added.

Caution – `datetwo` instructs IMTA to remove the leading two digits from four-digit dates. This is intended to provide compatibility with incompliant mail systems that require two digit dates; *it should never be used for any other purpose*.

Day of Week in Date Specifications (`dayofweek`, `nodayofweek`)

The RFC 822 specification allows for a leading day of the week specification in the date fields in message headers. However, some systems cannot accommodate day of the week information. This makes some systems reluctant to include this information, even though it is quite useful information to have in the headers.

The `dayofweek` and `nodayofweek` keywords control IMTA's processing of day of the week information. `dayofweek`, the default, instructs IMTA to retain any day of the week information and to add this information to date/time headers if it is missing.

Caution – `nodayofweek` instructs IMTA to remove any leading day of the week information from date/time headers. This is intended to provide compatibility with incompliant mail systems that cannot process this information properly; *it should never be used for any other purpose*.

Automatic Splitting of Long Header Lines (`maxheaderaddrs`, `maxheaderchars`)

Some message transfers, notably some sendmail implementations, cannot process long header lines properly. This often leads not just to damaged headers but to erroneous message rejection. Although this is a gross violation of standards, it is nevertheless a common problem.

IMTA provides per-channel facilities to split (break) long header lines into multiple, independent header lines. The `maxheaderaddrs` keyword controls how many addresses can appear on a single line. The `maxheaderchars` keyword controls how many characters can appear on a single line. Both keywords require a single integer parameter that specifies the associated limit. By default, no limit is imposed on the length of a header line nor on the number of addresses that can appear.

Header alignment and folding (`headerlabelalign`, `headerlinelength`)

The `headerlabelalign` keyword controls the alignment point for message headers enqueued on this channel; it takes an integer-valued argument. The alignment point is the margin where the contents of headers are aligned. For example, sample headers with an alignment point of 10 might look like this:

```
To:       joe@xyz.com
From:     mary@xyz.com
Subject:  Alignment test
```

The default `headerlabelalign` is 0, which causes headers not to be aligned. The `headerlinelength` keyword controls the length of message header lines enqueued on this channel. Lines longer than this are folded in accordance with RFC 822 folding rules.

These keywords only control the format of the headers of the message in the message queue; the actual display of headers is normally controlled by the user agent. In addition, headers are routinely reformatted as they are transferred across the Internet, so these keywords may have no visible effect even when used in conjunction with simple user agents that do not reformat message headers.

Automatic Defragmentation of Message/Partial Messages (`defragment`, `nodefragment`)–Enterprise Edition Only

The MIME standard provides the message/partial content type for breaking up messages into smaller parts. This is useful when messages have to traverse networks with size limits. Information is included in each part so that the message can be automatically reassembled after it arrives at its destination.

The `defragment` channel keyword and the defragmentation channel provide the means to reassemble messages in IMTA. When a channel is marked `defragment`, any message or partial messages queued to the channel are placed in the defragmentation channel queue instead. After all the parts have arrived, the message is rebuilt and sent on its way. The `nodefragment` disables this special processing. `nodefragment` is the default.

A defragment channel must be added to the IMTA configuration file in order for the `defragment` keyword to have any effect. If your configuration was built by the IMTA configuration utility, then you should already have such a channel.

Automatic Fragmentation of Large Messages (`maxblocks`, `maxlines`)

Some email systems or network transfers cannot handle messages that exceed certain size limits. IMTA provides facilities to impose such limits on a channel-by-channel basis. Messages larger than the set limits are automatically split (fragmented) into multiple, smaller messages. The `Content-type:` used for such fragments is `message/partial`, and a unique id parameter is added so that parts of the same message can be associated with one another and, possibly, be automatically reassembled by the receiving mailer.

The `maxblocks` and `maxlines` keywords are used to impose size limits beyond which automatic fragmentation are activated. Both of these keywords must be followed by a single integer value. `maxblocks` specifies the maximum number of blocks allowed in a message. An IMTA block is normally 1024 bytes; this can be changed with the `BLOCK_SIZE` option in the IMTA option file. `maxlines` specifies the maximum number of lines allowed in a message. These two limits can be imposed simultaneously, if necessary.

Message headers are, to a certain extent, included in the size of a message. Because message headers cannot be split into multiple messages, and yet they themselves can exceed the specified size limits, a rather complex mechanism is used to account for message header sizes. This logic is controlled by the `MAX_HEADER_BLOCK_USE` and `MAX_HEADER_LINE_USE` options in the IMTA option file.

MAX_HEADER_BLOCK_USE is used to specify a real number between 0 and 1. The default value is 0.5. A message's header is allowed to occupy this much of the total number of blocks a message can consume (specified by the maxblocks keyword). If the message header is larger, IMTA takes the product of MAX_HEADER_BLOCK_USE and maxblocks as the size of the header (the header size is taken to be the smaller of the actual header size and maxblocks) * MAX_HEADER_BLOCK_USE.

For example, if maxblocks is 10 and MAX_HEADER_BLOCK_USE is the default, 0.5, any message header larger than 5 blocks is treated as a 5-block header, and if the message is 5 or fewer blocks in size it is not fragmented. A value of 0 causes headers to be effectively ignored insofar as message-size limits are concerned.

A value of 1 allows headers to use up all of the size that's available. Each fragment always contains at least one message line, regardless of whether or not the limits are exceeded by this. MAX_HEADER_LINE_USE operates in a similar fashion in conjunction with the maxlines keyword.

Absolute Message Size Limits (blocklimit, linelimit)

Although fragmentation can automatically break messages into smaller pieces, it is appropriate in some cases to reject messages larger than some administratively-defined limit, (for example, to avoid service denial attacks). The blocklimit and linelimit keywords are used to impose absolute size limits. Each of these keywords must be followed by a single integer value.

blocklimit specifies the maximum number of blocks allowed in a message. IMTA rejects attempts to queue messages containing more blocks than this to the channel. An IMTA block is normally 1024 bytes; this can be changed with the BLOCK_SIZE option in the IMTA option file.

linelimit specifies the maximum number of lines allowed in a message. IMTA rejects attempts to queue messages containing more than this number of lines to the channel. These two limit blocklimit and linelimit, can be imposed simultaneously, if necessary.

IMTA options LINE_LIMIT and BLOCK_LIMIT can be used to impose similar limits on all channels. These limits have the advantage that they apply across all channels. Therefore, IMTA’s servers can make them known to mail clients prior to obtaining message recipient information. This simplifies the process of message rejection in some protocols.

Specify Maximum Length Header (`maxprocchars`)

Processing of long header lines containing lots of addresses can consume significant system resources. The `maxprocchars` keyword is used to specify the maximum length header that IMTA can process and rewrite. Messages with headers longer than this are still accepted and delivered; the only difference is that the long header lines are not rewritten in any way. A single integer argument is required. The default is processing headers of any length.

Message Logging (`logging`)

IMTA provides facilities for logging each message as it is enqueued and dequeued. All log entries are made to the file `mail.log_current` in the log directory `/var/opt/SUNWmail/imta/log/mail.log_current`. Logging is controlled on a per-channel basis. The `logging` keyword activates logging for a particular channel while the `nologging` keyword disables it.

Debugging Channel Master and Slave Programs (`master_debug`, `nomaster_debug`, `slave_debug`, `noslave_debug`)

Some channel programs include optional code to assist in debugging by producing additional diagnostic output. Two channel keywords are provided to enable generation of this debugging output on a per-channel basis. The keywords are `master_debug`, which enables debugging output in master programs, and `slave_debug`, which enables debugging output in slave programs. Both types of debugging output are disabled by default, corresponding to `nomaster_debug` and `noslave_debug`.

When activated, debugging output ends up in the log file associated with the channel program. The location of the log file may vary from program to program. Log files are usually kept in the IMTA log directory. Master programs usually have log file names of the form `x_master.log`, where `x` is the name of the channel; `slave` programs usually have log file names of the form `x_slave.log`. Also, some channel programs, notably TCP/IP and fax channel programs, may produce additional log files with names:

- `err_x_master.log`
- `err_x_slave.log`
- `di_x_master.log`

- di_x_xlave.log
- ph_x_master.log
- ph_x_slave.log

In the case of the local channel, master_debug enables debugging output when sending from the local channel, and slave_debug enables debugging output as messages are delivered to the local channel, with output usually appearing in the /var/opt/SUNWmail/imta/log/l_master.log.

Channel Keywords Table

The keywords following the channel name are used to assign various attributes to the channel. Keywords are case insensitive, and may be up to 32 characters long; any additional characters are ignored. The supported keywords are listed in TABLE 3-8; the keywords shown in **`boldface`** type are defaults.

TABLE 3-8 Channel Keywords Table

Keyword	Usage
`addrsperfile`	Number of addresses per message file
`addrsperjob`	Number of addresses to be processed by a single job
`after`	Specify time delay before master channel programs run
`allowswitchchannel`	Allow switching to this channel from a **`allowswitchchannel`** channel
`bangoverpercent`	Group `A!B%C` as `A!(B%C)`
`bidirectional`	Channel is served by both a master and slave program
`blocklimit`	Maximum number of IMTA blocks allowed per message
`cacheeverything`	Cache all connection information
`cachefailures`	Cache only connection failure information
`cachesuccess`	Cache only connection success information
`charset7`	Default character set to associate with 7-bit text messages
`charset8`	Default character set to associate with 8-bit text messages
`checkehlo`	Check the SMTP response banner for whether to use `EHLO`
`commentinc`	Leave comments in message header lines intact
`commentomit`	Remove comments from message header lines
`commentstrip`	Remove problematic characters from comment field in message header lines
`commenttotal`	Strip comments (material in parentheses) everywhere
`connectalias`	Do not rewrite addresses upon message dequeue
`connectcanonical`	Rewrite addresses upon message dequeue
`contchar`	Specify batch SMTP continuation line character
`contposition`	Specify folding point in batch SMTP lines
`convert_octet_stream`	Convert application or octet-stream material as appropriate
`copysendpost`	Send copies of failures to the postmaster unless the originator address is blank
`copywarnpost`	Send copies of warnings to the postmaster unless the originator address is blank
`daemon`	Specify the name of a gateway to which the daemon is routed
`datefour`	Convert date/time specifications to four-digit years
`datetwo`	Convert date/time specifications to two-digit years
`dayofweek`	Include day of week in date and time specifications
`defaultmx`	Channel determines whether or not to do MX lookups from network
`deferred`	Honor deferred delivery dates

TABLE 3-8 Channel Keywords Table

Keyword	Usage
`defragment`	Reassemble any MIME-compliant message/partial parts queued to this channel (Enterprise Edition only)
`domainvrfy`	Issue `SMTP VRFY` commands using full address
`ehlo`	Use EHLO on all initial SMTP connections
`eightbit`	Channel supports eight-bit characters
`eightnegotiate`	Channel should negotiate use of eight bit transmission, if possible
`eightstrict`	Channel should reject messages that contain unnegotiated eight-bit data
`errsendpost`	Send copies of failures to the postmaster if the originator address is illegal
`errwarnpost`	Send copies of warnings to the postmaster if the originator address is illegal
`expandlimit`	Process an incoming message “offline” when the number of addressees exceeds this limit
`exproute`	Explicit routing for this channel’s addresses
`filesperjob`	Number of queue entries to be processed by a single job
`headerinc`	Place the message header at the top of the message
`headerlabelalign`	Align headers
`headerlinelength`	Fold long headers
`headerread`	Apply header trimming rules from an options file to the message headers upon message enqueue (use with caution)
`headertrim`	Apply header trimming rules from an options file to the message headers (use with caution)
`holdexquota`	Hold extra messages for users who have exceeded their message quota
`identnone`	Do not perform IDENT lookups; do perform IP-to-hostname translation
`identnonenumeric`	Do not perform IDENT lookups or IP-to-hostname translation
`identtcp`	Perform IDENT lookups on incoming SMTP connections and IP to hostname translation
`identtcpnumeric`	Perform IDENT lookups on incoming SMTP connections, but do not perform IP to hostname translation
`ignoreencoding`	Ignore Encoding: header on incoming messages
`immediate`	Delivery started immediately after submission for messages of second-class or higher priority
`immnonurgent`	Delivery started immediately after submission, even for messages with lower-than-normal priority
`immnormal`	Delivery started immediately after submission for messages of normal-or-higher priority
`immurgent`	Delivery started immediately after submission for urgent messages only
`improute`	Implicit routing for this channel’s addresses
`inner`	Rewrite inner message headers
`innertrim`	Apply header trimming rules from an options file to inner message headers (use with caution)

TABLE 3-8 Channel Keywords Table

Keyword	Usage
`interpretencoding`	Interpret Encoding: header on incoming messages
`lastresort`	Specify a last-resort host
`linelength`	Message lines exceeding this length limit are wrapped
`linelimit`	Maximum number of lines allowed per message
`localvrfy`	Issue SMTP VRFY command using local address
`logging`	Log message enqueues and dequeues into the log file
`master`	Channel is served only by a master program
`master_debug`	Generate debugging output in the channel's master program output
`maxblocks`	Maximum number of IMTA blocks per message; longer messages are broken into multiple messages
`maxheaderaddrs`	Maximum number of addresses per message header line; longer header lines are broken into multiple header lines
`maxheaderchars`	Maximum number of characters per message header line; longer header lines are broken into multiple header lines
`maxjobs`	Maximum number of jobs that can be created at one time
`maxlines`	Maximum number of message lines per message; longer messages are broken into multiple messages
`maxprocchars`	Specify maximum length of headers to process
`multiple`	Accepts multiple destination hosts in a single message copy
`mx`	TCP/IP network and software supports MX record lookups
`nobangoverpercent`	Group `A!B%C` as `(A!B)%C` (default)
`nocache`	Do not cache any connection information
`noconvert_octet_stream`	Do not convert application/octet-stream material
`nodayofweek`	Remove day of week from date/time specifications
`nodeferred`	Do not honor deferred delivery dates
`nodefragment`	Do not perform special processing for message/partial messages
`noehlo`	Never use the SMTP EHLO command
`noexproute`	No explicit routing for this channel's addresses
`noexquota`	Return to originator any messages to users who are over quota
`noheaderread`	Do not apply header trimming rules from option file upon message enqueue
`noheadertrim`	Do not apply header trimming rules from options file
`noimproute`	No implicit routing for this channel's addresses
`noinner`	Do not rewrite inner message headers
`noinnertrim`	Do not apply header trimming to inner message headers
`nologging`	Do not log message enqueues and dequeues into the log file
`nomaster_debug`	Do not generate debugging output in the channel's master program output
`nomx`	TCP/IP network does not support MX lookups
`nonrandommx`	Do MX lookups; do not randomize returned entries with equal precedence
`nonurgentblocklimit`	Force messages above this size to wait unconditionally for a periodic job

TABLE 3-8 Channel Keywords Table

Keyword	Usage
`noreceivedfor`	Do not include envelope to address in `Received:` header
`noremotehost`	Use local host’s domain name as the default domain name to complete addresses
`norestricted`	Do not apply RFC 1137 restricted encoding to addresses
`noreverse`	Do not apply reverse database to addresses
`normalblocklimit`	Force messages above this size to nonurgent priority
`nosendpost`	Do not send copies of failures to the postmaster
`noslave_debug`	Do not generate debugging output in the channel’s slave program output
`nosmtp`	Channel does not use SMTP
`noswitchchannel`	Stay with the server channel; do not switch to the channel associated with the originating host; do not permit being switched to
`notices`	Specify the amount of time that may elapse before notices are sent and messages returned
`novrfy`	Do not issue `SMTP VRFY` commands
`nowarnpost`	Do not send copies of warnings to the postmaster
`nox_env_to`	Do not add `X-Envelope-to:` header lines while enqueuing
`percents`	Use % routing in the envelope; synonymous with 733
`period`	Specify periodicity of periodic channel service
`periodic`	Channel is serviced only periodically; immediate delivery processing is never done
`personalinc`	Leave personal names in message header lines intact
`personalomit`	Remove personal name fields from message header lines
`personalstrip`	Strip problematic characters from personal name fields in message header lines
`port`	Send to the specified TCP/IP port
`postheadbody`	Both the message’s header and body are sent to the postmaster when a delivery failure occurs
`postheadonly`	Only the message’s header is sent to the postmaster when a delivery failure occurs
`queue`	Specify queue master channel programs run in
`randommx`	Do MX lookups; randomize returned entries with equal precedence
`receivedfor`	Include envelope to address in `Received:` header
`remotehost`	Use remote host's name as the default domain name to complete addresses
`restricted`	Apply RFC 1137 restricted encoding to addresses
`returnenvelope`	Control use of blank envelope return addresses
`reverse`	Apply reverse database to addresses
`sendpost`	Send copies of failures to the postmaster
`sevenbit`	Channel does not support eight-bit characters; eight-bit characters must be encoded
`single`	Only one envelope `To:` address per message copy
`single_sys`	Each message copy must be for a single destination system

TABLE 3-8 Channel Keywords Table

Keyword	Usage
`slave`	Channel is serviced only by a slave program
`slave_debug`	Generate debugging output in the channel's slave program output
`smtp`	Channel uses SMTP
`smtp_cr`	Accept CR as an SMTP line terminator
`smtp_crlf`	Require CRLF as the SMTP line terminator
`smtp_lf`	Accept LF as an SMTP line terminator
`sourceroute`	Use source routes in the `message:` envelope; synonymous with 822
`subdirs`	Use multiple subdirectories
`switchchannel`	Switch from the server channel to the channel associated with the originating host
`threaddepth`	Number of messages triggering new thread with multithreaded SMTP client
`unrestricted`	Do not apply RFC 1137 restricted encoding to addresses
`urgentblocklimit`	Force messages above this size to normal priority
`user`	Use local usernames as user tags when possible
`usereplyto`	Specify mapping of `Reply-to:` header
`useresent`	Specify mapping of Resent- headers for non-RFC 822 environments
`warnpost`	Send copies of warnings to the postmaster
`x_env_to`	Add `X-Envelope-to:` header lines while enqueuing

Specifying a keyword not on this list is not an error (although it may be incorrect). On UNIX systems, undefined keywords are interpreted as group ids. The `imta test -rewrite` utility tells you if you have any keywords in your configuration file that don’t match a known rights list identifier.

Default `imta.cnf` File

The following is a default IMTA configuration file (`imta.cnf`) for a system behind the firewall (`mailhost.eng.company.com`) which has a routability scope of the mail server domain (`eng.company.com`). This particular file is available in the Enterprise Edition of SIMS.

```
! VERSION=1.1
! Modified by SIMS administration server on: Wed Aug 05 14:22:14
PDT 1998
!
! IMTA configuration file
!
! part I : rewrite rules
!
! rules to select local users
mailhost.eng.company.com
$U%mailhost.eng.company.com@mailhost.eng.company.com
mailhost.Eng $U%mailhost.eng.company.com@mailhost.eng.company.com
mailhost $U%mailhost.eng.company.com@mailhost.eng.company.com
eng.company.com $E$U%$D@mailhost.eng.company.com
!
! sims-ms
sims-ms.mailhost.eng.company.com $E$U@sims-ms-daemon
sims-ms.mailhost $E$U@sims-ms-daemon
!
! pipe
pipe.mailhost.eng.company.com $E$U@pipe-daemon
pipe.mailhost $E$U@pipe-daemon
!
! tcp_intranet
.eng.company.com $E$U%$H.eng.company.com@tcp_local-daemon
* $U%$&0.eng.company.com@tcp_local-daemon
eng $U%eng.company.com@tcp_local-daemon
.eng $U%$H.eng.company.com@tcp_local-daemon
!
! tcp_default_router
</etc/opt/SUNWmail/imta/internet.rules
. $E$U%$H@tcp-daemon
[] $E$U%$[IMTA_LIBUTIL,hostName,$L]@tcp-daemon
!
```

```
! reprocess
reprocess
$E$U%reprocess.mailhost.eng.company.com@reprocess-daemon
reprocess.mailhost.eng.company.com
$E$U%reprocess.mailhost.eng.company.com@reprocess-daemon
!
! defragment
defragment
$E$U%defragment.mailhost.eng.company.com@defragment-daemon
defragment.mailhost.eng.company.com
$E$U%defragment.mailhost.eng.company.com@defragment-daemon
!
! conversion
conversion
$E$U%conversion.mailhost.eng.company.com@conversion-daemon
conversion.mailhost.eng.company.com
$E$U%conversion.mailhost.eng.company.com@conversion-daemon
!
! bitbucket
bitbucket
$E$U%bitbucket.mailhost.eng.company.com@bitbucket-daemon
bitbucket.mailhost.eng.company.com
$E$U%bitbucket.mailhost.eng.company.com@bitbucket-daemon

!
! part II : channel blocks
!
! delivery channel to local /var/mail store
l noswitchchannel copywarnpost copysendpost postheadonly charset7
us-ascii charset8 iso-8859-1 subdirs 20 immnonurgent logging
viaaliasrequired notices 1 2 4 7 serviceall
mailhost.eng.company.com

!
! sims-ms
sims-ms queue single_job copywarnpost copysendpost postheadonly
noswitchchannel charset7 us-ascii charset8 iso-8859-1 subdirs 20
immnonurgent logging serviceall
sims-ms-daemon

!
! pipe
pipe single subdirs 20 copywarnpost copysendpost postheadonly
immnonurgent noswitchchannel logging notices 1 2 4 7 serviceall
pipe-daemon
```

```
!
! tcp_intranet
tcp_local smtp single_sys subdirs 20 copywarnpost copysendpost
postheadonly immnonurgent noreverse logging notices 1 2 4 7
tcp_local-daemon mailhost.eng.company.com
tcp-daemon mailhost.eng.company.com
!
! reprocess
reprocess copywarnpost copysendpost postheadonly
reprocess-daemon
!
! tcp_default_router
tcp_default_router smtp daemon mailhost.eng.company.com
copysendpost copywarnpost postheadonly subdirs 20 immnonurgent
logging notices 1 2 4 7
!
! defragment
defragment copywarnpost copysendpost postheadonly
defragment-daemon
!
! conversion
conversion copywarnpost copysendpost postheadonly
conversion-daemon

!
! bitbucket
bitbucket copywarnpost copysendpost postheadonly
bitbucket-daemon
```

The `imta.cnf` file defines several channels. The default channels defined in the sample default `imta.cnf` file are described in TABLE 3-9.

TABLE 3-9 `imta.cnf` Channel Descriptions

channel	description
`l`	The local (`l`) channel is used to deliver messages to addresses on the local host. Message files queued to the l channel are delivered to local users by the local channel program `l_master`. `/opt/SUNWmail/imta/bin/sendmail` is the slave program invoked to queue the message to the appropriate queues.
`sims-ms`	The `sims-ms` channel is used to deliver messages to the SIMS Message Store. Message files queued to this channel are delivered by the `ims_master` program

TABLE 3-9 `imta.cnf` Channel Descriptions

channel	description
`pipe`	Pipe channels are used to perform delivery via a site supplied program or script. Commands executed by the `pipe` channel are controlled by the administrator via the `imta program` interface. Pipe channels are also used by the autoreply program.

`tcp_intranet`
`tcp_local`
`tcp_default_router`
	These channels implement SMTP over TCP/IP. The multithreaded TCP SMTP channel includes a multithreaded SMTP server that runs under the control of the IMTA SMTP Dispatcher. Outgoing SMTP mail is processed by the channel program `tcp_smtp_client`, and run as needed under the control of the IMTA Job Controller.

`reprocess`	The `reprocess` channel is the intersection of all other channel programs—they perform only those operations that are shared with other channels. It is simply a channel queue whose contents are processed and requeued to other channels.
`defragment`	The `defragment` channel provides the means to reassemble messages.
`conversion`	The `conversion` channel performs body-part-by-body-part conversions on messages flowing through the IMTA.
`bitbucket`	The `bitbucket` channel is used for messages that need to be discarded.
`addressing`	The `addressing` channel is used to extract addressing information from a message body.

Address Rewrite Example

The example in this section takes a mail message and tracks it through the rewrite rules.

1. A mail message arrives for `jdoe@eng.company.com`.
2. The `imta.cnf` file is scanned to find a match for the domain part of the address. If it matches any of the rules in the first rewrite rule section (rules to select local user or `l` channel), the user is looked up in the alias database. In this example the address domain part matches rule four in the first section of rewrite rules.
3. The alias cache is searched for the `jdoe` entry. The `imta.cnf` file is again scanned to find a match with the domain part of the address returned by the alias database search.

   The address may be returned as one of the following:

- If the address returned is `jdoe@sims-ms.myhost.eng.company.com`, the domain part matches a rule whose routing system is that of the message store channel. The message is then queued to the `sims-ms` channel and delivered to the store by the channel's master program.
- If the address returned is `jdoe@myhost.eng.company.com`, the domain part matches a rule in the `l` channel section. If the address matches the address previously looked up in the alias table, the destination channel is set to `l` (`/var/mail`) and the message is enqueued. If it does not match, the same operation is reiterated until it stabilizes on one address, the limit of the alias lookup is reached, or a self reference (a:b, b:c, c:a) is found. In the last two cases, the address is not resolved, which causes the messages to be bounced (unknown user error message). The alias lookup default is 10. This value can be changed by configuring the `MAX_ALIAS_LEVELS` option in the option file.
- If the address returned is `jdoe@host2.eng.company.com`, the domain part matches a rule in the Intranet SMTP channel section and the message is enqueued to the `tcp_local` channel (intranet channel) and so on.
- If the string `jdoe` does not match anything in the alias cache, the message is returned to the originator with the "user unknown" error.

Local Channel

The local channel (l) is used to deliver messages to addresses on the local host.

When using a mail user agent on the local system to send mail (to anywhere), the `sendmail` utility (`/opt/SUNWmail/imta/bin/sendmail`) is invoked as the replacement for sendmail to queue the messages to the appropriate queues, and then the channel programs for those queues will process the messages.

The IMTA configuration utility always generates a local channel.

The Local Channel Option File

An option file may be used to control various characteristics of the local channel. This local channel option file must be stored in the IMTA configuration directory and named `l_option` (for example, `/etc/opt/SUNWmail/imta/l_option`).

Option files consist of several lines. Each line contains the setting for one option. An option setting has the form:

```
option=value
```

value may be either a string or an integer, depending on the option's requirements. If the option accepts an integer value a base may be specified using notation of the form *b%v*, where *b* is the base expressed in base 10 and *v* is the actual value expressed in base *b*.

SMTP Channel Option Files

An option file may be used to control various characteristics of TCP/IP channels. Such an option file must be stored in the IMTA configuration directory (`/etc/opt/SUNWmail/imta`) and named *`x`*`_option`, where *`x`* is the name of the channel.

Format of the File

Option files consist of several lines. Each line contains the setting for one option. An option setting has the form:

```
option=value
```

`value` may be either a string or an integer, depending on the option's requirements. If the option accepts an integer value, a base may be specified using notation of the form *b%v*, where *b* is the base expressed in base 10 and *vb*.

Available SMTP Channel Options

The available options are listed in TABLE 3-10.

TABLE 3-10 SMTP Channel Options

Option	Description
ALLOW_TRANSACTIONS_PER_SESSION (Integer)	Set a limit on the number of messages allowed per connection. The default is no limit.
ALLOW_RECIPIENTS_PER_TRANSACTION (Integer)	Set a limit on the number of recipients allowed per message. The default is no limit.
ATTEMPT_TRANSACTIONS_PER_SESSION (Integer)	Set a limit on the number of messages IMTA will attempt to transfer during any one connection session.
COMMAND_RECEIVE_TIME (Integer)	This option specifies, in minutes, how long to wait to receive general SMTP commands (commands other than those with explicitly specified time-out values set using other specifically named options).
COMMAND_TRANSMIT_TIME (Integer)	This option specifies, in minutes, how long to spend transmitting general SMTP commands (commands other than those with explicitly specified time-out values set using other specifically named options).
CONTINUATION_CHARS (List of Integers)	Additional characters that act like the continuation character specified using the `contchar` channel keyword. This is a list of integer values.
DATA_RECEIVE_TIME (Integer)	This option specifies, in minutes, how long to wait to receive data during an SMTP dialogue. The default is 60.
DATA_TRANSMIT_TIME (Integer)	This option specifies, in minutes, how long to spend transmitting data during an SMTP dialogue. The default is 10.
DISABLE_ADDRESS (0 or 1)	The IMTA SMTP server implements a private command XADR. This command returns information about how an address is routed internally by IMTA as well as general channel information. Releasing such information may constitute a breach of security for some sites. Setting the DISABLE_ADDRESS option to 1 disables the XADR command. The default is 0, which enables the XADR command.

TABLE 3-10 SMTP Channel Options

Option	Description
DISABLE_EXPAND (0 or 1)	The SMTP EXPN command is used to expand mailing lists. Exposing the contents of mailing lists to outside scrutiny may constitute a breach of security for some sites. The DISABLE_EXPAND option, when set to 1, disables the EXPN command completely. The default value is 0, which causes the EXPN command to work normally. Note that mailing list expansion can also be blocked on a list-by-list basis by setting the expandable attribute to False in the list's directory entry.
DISABLE_STATUS (0 or 1)	The IMTA SMTP server implements a private command XSTA. This command returns status information about the number of messages processed and currently in the IMTA channel queues. Releasing such information may consisted a breach of security for some sites. Setting the DISABLE_STATUS option to 1 disables the XSTA command. The default is 0, which enables the XSTA command.
DOT_TRANSMIT_TIME (Integer)	This option specifies, in minutes, how long to spend transmitting the dot (.) terminating the data in an SMTP dialogue. The default is 10.
HIDE_VERIFY (0 or 1)	The SMTP VRFY command can be used to establish the legality of an address before using it. Unfortunately this command has been abused by automated query engines in some cases. The HIDE_VERIFY option, when set to 1, tells IMTA not to return any useful information in the VRFY command result. The default value is 0, which causes VRFY to act normally.
MAIL_TRANSMIT_TIME (Integer)	This option specifies, in minutes, how long to spend transmitting the SMTP command MAIL FROM:. The default is 10.
MAX_CLIENT_THREADS	An integer number indicating the maximum number of simultaneous outbound connections that the client channel program will allow. Note that multiple processes may be used for outbound connections, depending on how you have channel-processing queues setup. This option controls the number of threads per process. The default if this option is not specified is 10.

TABLE 3-10 SMTP Channel Options

Option	Description
MAX_SERVER_THREADS	An integer number indicating the maximum number of simultaneous inbound connections that the SMTP server program will allow. Note that since only one server process is allowed, this option effectively controls the total number of simultaneous inbound SMTP connections IMTA can handle. The default if this option is not specified is 40.
RCPT_TRANSMIT_TIME (Integer)	This option specifies, in minutes, how long to spend transmitting the SMTP command RCPT TO:. The default is 10.
STATUS_DATA_RECEIVE_TIME (Integer)	This option specifies, in minutes, how long to wait to receive the SMTP response to your sent data--that is, how long to wait to receive a "550" (or other) response to the dot-terminating-sent data. The default value is 10. See also the STATUS_DATA_RECV_PER_ADDR_TIME, STATUS_DATA_RECV_PER_BLOCK_TIME, and STATUS_DATA_RECV_PER_ADDR_PER_BLOCK_ TIME options.
STATUS_DATA_RECV_PER_ADDR_TIME (Floating Point Value)	This option specifies an adjustment factor for how long to wait to receive the SMTP response to your sent data based on the number of addresses in the MAIL TO: command. This value is multiplied by the number of addresses and added to the base wait time (specified with the STATUS_DATA_RECV_TIME option). The default is 0.083333.
STATUS_DATA_RECV_PER_BLOCK_TIME (Floating Point Value)	This option specifies an adjustment factor for how long to wait to receive the SMTP response to your sent data based on the number of blocks sent. This value is multiplied by the number of blocks and added to the base wait time (specified with the STATUS_DATA_RECV_TIME option). The default is 0.001666.
STATUS_DATA_RECV_PER_ADDR_PER_BLOCK_TIME (Floating Point Value)	This option specifies an adjustment factor for how long to wait to receive the SMTP response to your sent data based on the number of addresses (in the MAIL TO: command) per number of blocks sent. This value is multiplied by the number of addresses per block and added to the base wait time (specified with the STATUS_DATA_RECV_TIME option). The default is 0.003333.

TABLE 3-10 SMTP Channel Options

Option	Description
`STATUS_MAIL_RECEIVE_TIME` (Integer)	This option specifies, in minutes, how long to wait to receive the SMTP response to a sent `MAIL FROM:` command. The default is 10.
`STATUS_RCPT_RECEIVE_TIME` (Integer)	This option specifies, in minutes, how long to wait to receive the SMTP response to a sent `RCPT TO:` command. The default value is 10.
`STATUS_RECEIVE_TIME` (Integer)	This option specifies, in minutes, how long to wait to receive a banner line upon connection. The default value is 10.
`STATUS_TRANSMIT_TIME` (Integer)	This option specifies, in minutes, how long to spend transmitting the banner line upon receiving a request to open an SMTP connection.

The Pipe Channel

The pipe channel is used to perform delivery via per-user site-supplied programs. It provides a similar functionality to `sendmail`'s pipe (`|`). The following differences are designed to not pose a security threat. First, delivery programs to be invoked by the pipe channel need to be "registered" by the system administrator. This registration is done performed via the `imta program` utility. See "imta program" on page 99 for information about `imta program`.

Second, unlike the `sendmail` pipe functionality, the IMTA pipe channel does not pipe the message to be processed to the program or script. Instead, it writes the message to be processed to a temporary file and then forks a subprocess to run the site-supplied command for that message. That command should make use of the name of the temporary file which can be substituted into the command by the channel. The temporary file should not be deleted or altered by the subprocess; the channel will delete the temporary file itself. If it is not possible to prevent the subprocess from disrupting the file, then the pipe channel should be marked with the `single` channel keyword.

Delivery programs invoked by the pipe channel must return meaningful error codes so that the channel knows whether to dequeue, deliver for later processing, or return messages.

If the subprocess exits with exit code of 0 (`EX_OK`), the message is presumed to have been delivered successfully and is removed from IMTA's queues. If it exits with an exit code of 71, 74, 75, or 79 (`EX_OSERR`, `EX_IOERR`, `EX_TEMPFAIL`, or `EX_DB`), a temporary error is presumed to have occurred and delivery of the message is deferred. If any other exit code is returned, then the message will be returned to its originator as undeliverable. These exit codes are defined in the system header file `sysexits.h`.

Using the Pipe Channel

The `imta program` utility gives a name to each UNIX command that the administrator registers as able to be invoked by the pipe channel. This name can then be used by the end user as a value of their `mailprogramdeliveryinfo` LDAP attribute in order to enable delivery via the command corresponding to this name. The attribute `maildeliveryoption` must have one value equal to `program`.

Conversion Channel

The conversion channel performs arbitrary body-part-by-body-part conversions on messages flowing through IMTA. Any subset of IMTA traffic can be selected for conversion and any set of programs or command procedures can be used to perform conversion processing. (IMTA's native conversion facilities are fairly limited, so the ability to call external converters is crucial.) A special conversion channel configuration is consulted to choose an appropriate conversion for each body part.

Selecting Traffic for Conversion Processing

Although conversion processing is done using a regular IMTA channel program, under normal circumstances this channel is never specified directly either in an address or in an IMTA rewrite rule. IMTA controls access to the conversion channel via the `CONVERSIONS` mapping table in the IMTA mappings file (`/etc/opt/SUNWmail/imta/mappings`).

As IMTA processes each message it probes the `CONVERSIONS` mapping (if one is present) with a string of the form:

```
IN-CHAN=source-channel;OUT-CHAN=destination-channel;CONVERT
```

source-channel is the channel from which the message is coming and *destination-channel* is the channel to which the message is heading. If the mapping produces a result, it should either be the string `Yes` or `No`. If `Yes` is produced, IMTA will divert the message from its regular destination to the conversion channel. If No is produced or if no match is found, the message will be queued to the regular destination channel.

For example, if all messages going to the tcp_intranet channel that do not originate from the tcp_intranet channel require conversion processing, the following mapping would then be appropriate:

```
CONVERSIONS

IN-CHAN=tcp_intranet;OUT-CHAN=tcp_intranet;CONVERT NO
IN-CHAN=*;OUT-CHAN=tcp_intranet;CONVERT YES
```

Configuration

Configuration of the conversion channel in the IMTA configuration file (`imta.cnf`) is performed by default. Address of the form `user@conversion.`*localhostname* or `user@conversion` will be routed through the conversion channel, regardless of what the `CONVERSIONS` mapping states.

Conversion Control

The actual conversions performed by the conversion channel are controlled by rules specified in the IMTA conversion file. This is the file specified by the `IMTA_CONVERSION_FILE` option in the IMTA tailor file. By default, this is the file `/etc/opt/SUNWmail/imta/conversions.`

The IMTA conversion file is a text file containing entries in a format that is modeled after MIME Content-Type: parameters. Each entry consists of one or more lines grouped together; each line contains one or more `name=`**`value;`** parameter clauses. Quoting rules conform to MIME conventions for Content-Type: header line parameters. Every line except the last must end with a semicolon (`;`). A physical line in this file is limited to 252 characters. You can split a logical line into multiple physical lines using the backslash (\) continuation character. Entries are terminated either by a line that does not end in a semicolon, one or more blank lines, or both. For example, the following entry specifies that `application/wordperfect5.1` parts in messages sent to the local channel should be converted to DDIF:

```
out-chan=l; in-type=application; in-subtype=wordperfect5.1;
out-type=application; out-subtype=ddif; out-mode=block;
command="CONVERT/DOCUMENT 'INPUT_FILE'/FORMAT=WORDP 'OUTPUT_FILE'/FORMAT=DDIF"
```

Available Parameters

The rule parameters currently provided are shown in TABLE 3-11. Parameters not listed in the table are ignored.

TABLE 3-11 Available Conversion Parameters

Parameter	Description
`COMMAND`	Command to execute to perform conversion. This parameter is required; if no command is specified, the entry is ignored.
`DELETE`	0 or 1. If this flag is set, the message part will be deleted. (If this is the only part in a message, then a single empty text part will be substituted.)
`IN-A1-FORMAT`	Input A1-Format from enclosing `MESSAGE/RFC822` part.
`IN-A1-TYPE`	Input A1-Type from enclosing `MESSAGE/RFC822` part.
`IN-CHAN`	Input channel to match for conversion (wildcards allowed). The conversion specified by this entry will only be performed if the message is coming from the specified channel.
`IN-CHANNEL`	Synonym for `IN-CHAN`.
`IN-DESCRIPTION`	Input MIME Content-Description.
`IN-DISPOSITION`	Input MIME Content-Disposition.
`IN-DPARAMETER-DEFAULT-`*n*	Input MIME Content-Disposition parameter value default if parameter is not present. This value is used as a default for the `IN-DPARAMETER-VALUE-`*n* test when no such parameter is specified in the body part.
`IN-DPARAMETER-NAME-`*n*	Input MIME Content-Disposition parameter name whose value is to be checked; *n* = 0, 1, 2, ...
`IN-DPARAMETER-VALUE-`*n*	Input MIME Content-Disposition parameter value that must match corresponding `IN-DPARAMETER-NAME` (wildcards allowed). The conversion specified by this entry is only performed if this field matches the corresponding parameter in the body part's Content-Disposition: parameter list.
`IN-PARAMETER-DEFAULT-`*n*	Input MIME Content-Type parameter value default if parameter is not present. This value is used as a default for the `IN-PARAMETER-VALUE-`*n* test when no such parameter is specified in the body part.

TABLE 3-11 Available Conversion Parameters

Parameter	Description
`IN-PARAMETER-NAME-`*n*	Input MIME Content-Type parameter name whose value is to be checked; *n* = 0, 1, 2, ...
`IN-PARAMETER-VALUE-`*n*	Input MIME Content-Type parameter value that must match corresponding `IN-PARAMETER-NAME` (wildcards allowed). The conversion specified by this entry is performed only if this field matches the corresponding parameter in the body part's Content-Type: parameter list.
`IN-SUBJECT`	Input Subject from enclosing `MESSAGE/RFC822` part.
`IN-SUBTYPE`	Input MIME subtype to match for conversion (wildcards allowed). The conversion specified by this entry is performed only if this field matches the MIME subtype of the body part.
`IN-TYPE`	Input MIME type to match for conversion (wildcards allowed). The conversion specified by this entry is only performed if this field matches the MIME type of the body part.
`ORIGINAL-HEADER-FILE`	0 or 1. If set to 1, the original headers or the enclosing `MESSAGE/RFC822` part are written to the file represented by the `OUTPUT_HEADERS` symbol.
`OUT-A1-FORMAT`	Output A1-Format.
`OUT-A1-TYPE`	Output A1-Type.
`OUT-CHAN`	Output channel to match for conversion (wildcards allowed). The conversion specified by this entry will be performed only if the message is destined for the specified channel.
`OUT-CHANNEL`	Synonym for `OUT-CHAN`.
`OUT-DESCRIPTION`	Output MIME Content-Description if it is different than the input MIME Content-Description.
`OUT-DISPOSITION`	Output MIME Content-Disposition if it is different than the input MIME Content-Disposition.
`OUT-DPARAMETER-NAME-`*n*	Output MIME Content-Disposition parameter name; *n*=0, 1, 2, ...
`OUT-DPARAMETER-VALUE-`*n*	Output MIME Content-Disposition parameter value corresponding to `OUT-DPARAMETER-NAME-`*n*.

TABLE 3-11 Available Conversion Parameters

Parameter	Description
OUT-MODE	Mode in which to read the converted file. This should be one of: BLOCK, RECORD, RECORD-ATTRIBUTE, TEXT.
OUT-ENCODING	Encoding to apply to the converted file.
OUT-PARAMETER-NAME-*n*	Output MIME Content-Type parameter name; *n* = 0, 1, 2, ...
OUT-PARAMETER-VALUE-*n*	Output MIME Content-Type parameter value corresponding to OUT-PARAMETER-NAME-*n*.
OUT-SUBTYPE	Output MIME type if it is different than the input MIME type.
OUT-TYPE	Output MIME type if it is different than the input type.
OVERRIDE-HEADER-FILE	0 or 1. If set, then headers are read from the OUTPUT_HEADERS symbol, overriding the original headers in the enclosing MESSAGE/RFC822 part.
PARAMETER-SYMBOL-*n*	Content-Type parameters to convert to environment variables if present; *n* = 0, 1, 2, ... Takes as argument the name of the MIME parameter to convert, as matched by an IN-PARAMETER-NAME-*n* clause. Each PARAMETER-SYMBOL-*n* is extracted from the Content-Type: parameter list and placed in an environment variable of the same name prior to executing the converter.
PARAMETER-COPY-*n*	A list of the Content-Type: parameters to copy from the input body part's Content-Type: parameter list to the output body part's Content-Type: parameter list; *n*=0, 1, 2, ... Takes as argument the name of the MIME parameter to copy, as matched by an IN-PARAMETER-NAME-*n* clause.
PART-NUMBER	Dotted integers: *a. b. c*... The part number of the MIME body part.
RELABEL	0 or 1. This flag is ignored during conversion channel processing.

Predefined Environment Variables

TABLE 3-12 shows the basic set of environment variables available for use by the conversion command.

TABLE 3-12 Environment Variables used by Conversion Channel

Environment Variable	Description
INPUT_TYPE	The content type of the input message part.
INPUT_SUBTYPE	The content subtype of the input message part.
INPUT_DESCRIPTION	The content description of the input message part.
INPUT_DISPOSITION	The content disposition of the input message part.
OUTPUT_FILE	The name of the file where the converter should store its output. The converter should create and write this file.
OUTPUT_FILE	The name of the file where the converter should store headers for an enclosing MESSAGE/RFC822 part. The converter should create and write this file.

Additional environment variables containing Content-Type: information can be created as they are needed using the PARAMETER-SYMBOL-*n* facility.

Conversion Entry Scanning and Application

The conversion channel processes each message part-by-part. The header of each part is read and its Content-Type: and other header information is extracted. The entries in the conversion file are then scanned in order from first to last; any IN-parameters present and the OUT-CHAN parameter, if present, are checked. If all of these parameters match the corresponding information for the body part being processed, then the conversion specified by the remainder of the parameter is performed.

More specifically, the matching checks: if the IN-CHAN and OUT-CHAN parameters match the channels through which the message is passing; and if the PART-NUMBER matches the structured part number2 of the message part; and if all of the IN-CHAN, IN-PARAMETER-NAME, IN-PARAMETER-VALUE, IN-SUBTYPE, and IN-TYPE, parameters match the Content-Type: of the message; and if all of the IN-DISPOSITION, IN-DPARAMETER-NAME, and IN-DPARAMETER-VALUE parameters match the Content-Disposition of the message; and if the IN-DESCRIPTION matches the Content-Description of the message; and if the IN-SUBJECT, IN-A1-TYPE, and IN-A1-FORMAT of the headers of the immediately enclosing message (MESSAGE/RFC822 part) match those immediately enclosing the message part. Only if all specified parameters match is the entry consider to match. Scanning terminates once a matching entry has been found or all entries have been exhausted. If no entry matches no conversion is performed.

If the matching entry specifies DELETE=1, then the message part is deleted. Otherwise, the command specified by the COMMAND parameter is executed.

Once an entry with a COMMAND parameter has been selected the body part is extracted to a file. The converter execution environment is prepared as specified by the PARAMETER-SYMBOL-*n* parameters. Finally, a subprocess is created to run the command specified by the COMMAND parameter. The command should perform the necessary conversion operation, reading the file specified by the INPUT_FILE environment variable and producing the file specified by the OUTPUT_FILE environment variable.

Conversion operations are terminated and no conversion is performed if the forked command returns an error.

If the command succeeds, the resulting output file is read as specified by the OUT-MODE parameter and a new body part containing the converted material is constructed according to the OUT-ENCODING, OUT-PARAMETER-NAME-*n*, OUT-PARAMETER-VALUE-*n*, OUT-SUBTYPE, OUT-TYPE, OUT-DESCRIPTION, OUT-DISPOSITION, and OUT-DPARAMETER-VALUE-*n* parameters.

This process is repeated for each part of the message until all parts have been processed.

Headers in an Enclosing MESSAGE/RFC822 Part

When performing conversions on a message part, the conversion channel has access to the headers in an enclosing MESSAGE/RFC822 part, or to the message headers if there is no enclosing MESSAGE/RFC822 part.

For instance, the IN-A1-TYPE and IN-A1-FORMAT parameters can be used to check the A1-Type and A1-Format headers of an enclosing part, and the OUT-A1-TYPE and OUT-A1-FORMAT parameters can be used to set those enclosing headers.

More generally, if an entry is selected that has ORIGINAL-HEADER-FILE=1, then all the original headers of the enclosing MESSAGE/RFC822 part are written to the file represented by the OUTPUT_HEADERS environment variable. If OVERRIDE-HEADER-FILE=1, then the conversion channel will read and use as the headers on that enclosing part the contents of the file represented by the OUTPUT_HEADERS environment variable.

Environment Variable Substitution in Conversion Entries

Environment variable names may be substituted into a conversion entry by enclosing the name in single quotes. For instance, with a site supplied command procedure CONVERTER that attempts to perform various conversions and which defines OUTPUT_TYPE and OUTPUT_SYMBOL job logicals describing its output, one might use an entry along the lines of:

```
in-chan=tcp_local; out-chan=l; in-type=application; in-subtype=*;
out-type='OUTPUT_TYPE'; out-subtype='OUTPUT_SUBTYPE';
command="@CONVERTER 'INPUT_FILE' 'OUTPUT_FILE' 'INPUT_TYPE' 'INPUT_SUBTYPE'"
```

To obtain a literal single quote in a conversion entry, quote it with the backslash character, \'. To obtain a literal

backslash in a conversion entry, use two backslashes, \\.

Calling Out to a Mapping Table from a Conversion Entry

The value for a conversion parameter may be obtained by calling out to a mapping table. The syntax for calling out to a mapping table is as follows:

```
'mapping-table-name:mapping-input'
```

Consider the following mapping table:

```
X-ATT-NAMES
postscript          PS.PS
wordperfect5.1      WPC.WPC
msword              DOC.DOC
```

The following conversion entry for the above mapping table results in substituting generic file names in place of specific file names on attachments:

```
out-chan=tcp_local; in-type=application; in-subtype=*;
in-parameter-name-0=name; in-parameter-value-0=*:[*]*;
  out-type=application; out-subtype='INPUT-SUBTYPE';
  out-parameter-name-0=name;
  out-parameter-value-0='X-ATT-NAMES:\'INPUT_SUBTYPE\''
  command="COPY 'INPUT_FILE' 'OUTPUT_FILE'"
```

UUCP Channel

Note – The UUCP channel is available only in the Enterprise edition of SIMS.

UUCP (UNIX to UNIX Copy Program) is an asynchronous terminal line-based system providing support for file transfer and remote execution between different computer systems. These primitive operations are then used to construct a mail system, which is also, confusingly, known as UUCP.

Solaris supports the HoneyDanBer version of UUCP. Refer to the book *Configuring Your Network Software* for information on setting up UUCP on your system.

The UUCP channel is not one of the default channels. It cannot be configured through the Administration Console. This section describes how to set up the UUCP channel by editing the IMTA configuration file, `imta.cnf`.

Setting up the Channel

Two or more channels are needed for the IMTA to communicate using UUCP. A single common channel is used for all incoming messages, no matter from what system they originated. An additional outbound channel is needed for each system connected via UUCP. The incoming message channel is slave-only and should never have any messages queued to it. The outgoing message channels are master-only.

Adding the Channel to the `imta.cnf` File

The entry for the incoming message channel should resemble the following (do not use a different channel name):

```
uucp_gateway uucp slave
uucp-gateway
```

Entries for outgoing UUCP message channels will vary depending on the name of the system to which the channel connects. For example, suppose the remote system's official name is `uuhost.bravo.com` and its UUCP name is simply `uuhost`. A channel definition for this system might be:

```
uucp_uuhost uucp master
uuhost-uucp
uuhost.bravo.com uuhost
```

In this case, the name of the remote host to which the channel connects is derived from the channel name. When a second channel connecting to the same remote host is needed, it can be defined as:

```
uucp_second uucp master daemon uuhost
uuhost-second
uuhost.bravo.com uuhost
```

In this case, the daemon channel keyword has been used to explicitly specify the name of the remote system to which the channel connects.

If the official name and UUCP name are the same, `ymir`, the entry can be simplified:

```
uucp_uuhost uucp master
uuhost
```

Rewrite rules should be set up to point at the proper outgoing channel using the channel's official host name. For example

```
uucp.ymir.university.edu $E$U@ymir
```

Setting up the Master Program

Once the UUCP channels have been added to the configuration file, the UUCP master program should be ready to use. No additional log, script or option files are needed.

Setting up the Slave Program

The IMTA uucp_slave program is used to replace the rmail program on UNIX. You should rename the original rmail program (for example, to rmail.org) and create a symbolic link that links rmail to /opt/SUNWmail/imta/lib/uucp_slave as follows:

```
# cd /usr/bin
# mv rmail rmail.org
# ln -s /opt/SUNWmail/imta/lib/uucp_slave rmail
```

Log Files

Various log files are created during the operation of the UUCP channels. All IMTA-specific log files are kept in the IMTA log directory, (/var/opt/SUNWmail/imta/log).

While running, the uucp_master program creates a log file, x_master.logfile where x is the channel name. x_master.logfile logs each message as it is queued to the UUCP system.

Operation of the uucp_slave program creates a log file called rmail.logfile.

Returning Undelivered Messages

The IMTA automatically returns undeliverable messages after a certain amount of time has elapsed. However, UUCP maintains its own queues for files, so it is possible for messages to get stuck in the UUCP queues where the IMTA's regular message return job cannot see them.

Thus, an additional periodic cron job is needed to return undeliverable UUCP messages. This job operates in the same way as the IMTA's regular message return job except that it scans the UUCP queues and not the IMTA queues. This job is scheduled by the cron daemon.

Starting the Message Return `cron` Job

The UUCP message return job should be scheduled by `cron`. To submit commands to the `cron` daemon, first become administrator, `inetmail`:

```
# su inetmail
```

To edit the `crontab` entries, issue the command:

```
% crontab -e
```

Add an entry similar to the following:

```
30 1 * * * /opt/SUNWmail/imta/lib/return_uucp.sh
</var/opt/SUNWmail/imta/log/return_uucp.log-`/opt/SUNWmail/imta/lib/unique_id` 2>&1
```

The example entry shown above would be used to run the UUCP return job at 1:30 am and create the log file `/var/opt/SUNWmail/imta/log/return_uucp.log-uniqueid`, where `uniqueid` will be a unique string disambiguifying the file name, allowing for multiple versions of the file. The first value specifies the minutes after the hour, and the second value specifies the hour—you may wish to specify other values according to the needs of your site. You should use the `return_uucp` shell script as shown above, which itself calls the program `/var/opt/SUNWmail/imta/bin/return_uucp`, rather than the UUCP cleanup command, since `return_uucp` will honor the notices channel keyword and understand the MIME format of the messages.

The Mapping File

Many components of IMTA employ table lookup-oriented information. Generally speaking, this sort of table is used to transform (that is, map) an input string into an output string. Such tables, called mapping tables, are usually presented as two columns, the first (or left-hand) column giving the possible input strings and the second (or right-hand) column giving the resulting output string for the input it is associated with. Most of the IMTA databases are instances of just this sort of mapping table. IMTA database files, however, do not provide wildcard-lookup facilities, owing to inherent inefficiencies in having to scan the entire database for wildcard matches.

The mapping file provides IMTA with facilities for supporting multiple mapping tables. Full wildcard facilities are provided, and multistep and iterative mapping methods can be accommodated as well. This approach is more compute-intensive than using a database, especially when the number of entries is large. However, the attendant gain in flexibility may serve to eliminate the need for most of the entries in an equivalent database, and this may result in lower overhead overall.

The mapping file is used for reverse mapping, forward mapping, access control mapping, conversion mapping, etc.

Locating and Loading the Mapping File

All mappings are kept in the IMTA mapping file. (This is the file specified with the `IMTA_MAPPING_FILE` option in the IMTA tailor file; by default, this is `/etc/opt/SUNWmail/imta/mappings`.) The contents of the mapping file will be incorporated into the compiled configuration.

The mapping file should be world readable. Failure to allow world-read access will lead to erratic behavior.

File Format

The mapping file consists of a series of separate tables. Each table begins with its name. Names always have an alphabetic character in the first column. The table name is followed by a required blank line, and then by the entries in the table. Entries consist of zero or more indented lines. Each entry line consists of two columns separated by one or more spaces or tabs. Any spaces within an entry must be quoted. A blank line must appear after each mapping table name and between each mapping table; no blank lines can appear between entries in a single table. Comments are introduced by an exclamation mark (!) in the first column.

The resulting format looks like:

```
TABLE-1-NAME

   pattern1-1    template1-1
   pattern1-2    template1-2
   pattern1-3    template1-3
     .           .
     .           .
     .           .
   pattern1-n    template1-n

TABLE-2-NAME

   pattern2-1    template2-1
   pattern2-2    template2-2
   pattern2-3    template2-3
     .           .
     .           .
     .           .
   pattern2-n    template2-n

          .
          .
          .

TABLE-m-NAME

          .
          .
          .
```

An application using the mapping table `TABLE-2-NAME` would map the string `pattern2-2` into whatever is specified by `template2-2`. Each pattern or template can contain up to 252 characters. There is no limit to the number of entries that can appear in a mapping (although excessive numbers of entries may eat up huge amounts of CPU and can consume excessive amounts of memory). Long lines (over 252 characters) may be continued by ending them with a backslash (\). The white space between the two columns and before the first column may not be omitted.

Duplicate mapping table names are not allowed in the mapping file.

Including Other Files in the Mapping File

Other files may be included in the mapping file. This is done with a line of the form:

```
<file-spec
```

This will effectively substitute the contents of the file `file-spec` into the mapping file at the point where the include appears. The file specification should specify a full file path (device, directory, and so forth). All files included in this fashion must be world readable. Comments are also allowed in such included mapping files. Includes can be nested up to three levels deep. Include files are loaded at the same time the mapping file is loaded—they are not loaded on demand, so there is no performance or memory savings involved in using include files.

Mapping Operations

All mappings in the mapping file are applied in a consistent way. The only things that change from one mapping to the next is the source of input strings and what the output from the mapping is used for.

A mapping operation always starts off with an input string and a mapping table. The entries in the mapping table are scanned one at a time from top to bottom in the order in which they appear in the table. The left-hand side of each entry is used as pattern and the input string is compared in a case-blind fashion with that pattern.

Mapping Entry Patterns

Patterns can contain wildcard characters. In particular, the usual wildcard characters are allowed: an asterisk (*) will match zero or more characters and each percent sign (%) will match a single character. Asterisks, percent signs, spaces, and tabs can be quoted by preceding them with a dollar sign ($). Quoting an asterisk or percent sign robs it of any special meaning. Spaces and tabs must be quoted to prevent them from ending prematurely a pattern or template. Literal dollar sign characters should be doubled ($$), the first dollar sign quoting the second one.

All other characters in a pattern just represent and match themselves. In particular, single and double quote characters as well as parentheses have no special meaning in either mapping patterns or templates; they are just ordinary characters. This makes it easy to write entries that correspond to illegal addresses or partial addresses.

Asterisk wildcards maximize what they match by working from left to right across the pattern. For instance, when the string "a/b/c" is compared to the pattern */*, the left asterisk will match "a/b" and the right asterisk will match the remainder, "c".

Mapping Entry Templates

If the comparison of the pattern in a given entry fails, no action is taken; the scan proceeds to the next entry. If the comparison succeeds, the right-hand side of the entry is used as a template to produce an output string. The template effectively causes the replacement of the input string with the output string that is constructed from the instructions given by the template.

Almost all characters in the template simply produce themselves in the output. The one exception is a dollar sign ($).

A dollar sign followed by a dollar sign, space, or tab produces a dollar sign, space, or tab in the output string. Note that all these characters must be quoted in order to be inserted into the output string.

A dollar sign followed by a digit *n* calls for a substitution; a dollar sign followed by an alphabetic character is referred to as a "metacharacter". Metacharacters themselves will not appear in the output string produced by a template. See TABLE 3-13 for a list of the special substitution and standard processing metacharacters. Any other metacharacters are reserved for mapping-specific applications.

Note that any of the metacharacters $C, $E, $L, or $R, when present in the template of a matching pattern, will influence the mapping process, controlling whether it terminates or continues. That is, it is possible to set up iterative mapping table entries, where the output of one entry becomes the input of another entry. If the template of a matching pattern does not contain any of the metacharacters $C, $E, $L, or $R, then $E (immediate termination of the mapping process) is assumed.

The number of iterative passes through a mapping table is limited to prevent infinite loops. A counter is incremented each time a pass is restarted with a pattern that is the same length or longer than the previous pass. If the string has a shorter length than previously, the counter is reset to zero. A request to reiterate a mapping is not honored after the counter has exceeded 10.

TABLE 3-13 Mapping Template Substitutions and Metacharacters

Substitution sequence	Substitutes
`$n`	*n*th wildcarded field as counted from left to right starting from 0
`$#...#`	Sequence number substitution
`$|...|`	Apply specified mapping table to supplied string.

TABLE 3-13 Mapping Template Substitutions and Metacharacters

Substitution sequence	Substitutes
`${...}`	General database substitution.
`$[...]`	Invoke site-supplied routine; substitute in result.
Metacharacter	**Description**
`$C`	Continue the mapping process starting with the next table entry; use the output string of this entry as the new input string for the mapping process.
`$E`	End the mapping process now; use the output string from this entry as the final result of the mapping process.
`$L`	Continue the mapping process starting with the next table entry; use the output string of this entry as the new input string; after all entries in the table are exhausted, make one more pass, starting with the first table entry. A subsequent match may override this condition with a `$C`, `$E`, or `$R` metacharacter.
`$R`	Continue the mapping process starting with the first entry of the mapping table; use the output string of this entry as the new input string for the mapping process.
`$?x?`	Mapping entry succeeds x percent of the time.
`$\`	Force subsequent text to lowercase.
`$^`	Force subsequent text to uppercase.
`$_`	Leave subsequent text in its original case.

Wildcard Field Substitutions (`$n`)

A dollar sign followed by a digit `n` is replaced with the material that matched the *n*th wildcard in the pattern. The wildcards are numbered starting with 0. For example, the following entry would match the input string `PSI%A::B` and produce the resultant output string `b@a.psi.network.org`:

```
PSI$%*::*    $1@$0.psi.network.org
```

The input string `PSI%1234::USER` would also match producing `USER@1234.psi.network.org` as the output string. The input string `PSIABC::DEF` would not match the pattern in this entry and no action would be taken; that is, no output string would result from this entry.

Controlling Text Case (`$\`*,* `$^`*,* `$_`*)*

`$\` forces subsequent text to lowercase, `$^` forces subsequent text to uppercase, and `$_` causes subsequent text to retain its original case. For instance, these metacharacters may be useful when using mappings to transform addresses for which case is significant.

Processing Control (`$C`*,* `$L`*,* `$R`*,* `$E`*)*

The `$C`, `$L`, `$R`, and `$E` metacharacters influence the mapping process, controlling whether and when the mapping process terminates. `$C` causes the mapping process to continue with the next entry, using the output string of the current entry as the new input string for the mapping process. `$L` causes the mapping process to continue with the next entry, using the output string of the current entry as the new input string for the mapping process, and, if no matching entry is found, making one more pass through the table starting with the first table entry; a subsequent matching entry with a `$C`, `$E`, or `$R` metacharacter overrides this condition. `$R` causes the mapping process to continue from the first entry of the table, using the output string of the current entry as the new input string for the mapping process. `$E` causes the mapping process to terminate; the output string of this entry is the final output. `$E` is the default.

Mapping table templates are scanned left to right. So to set a `$C`, `$L`, or `$R` flag for entries that may "succeed" or "fail" (for example, general database substitutions or random-value controlled entries), put the `$C`, `$L`, or `$R` metacharacter to the left of the part of the entry that may succeed or fail; otherwise, if the remainder of the entry fails, the flag will not be seen.

Entry Randomly Succeeds or Fails (`$?`*x*`?`*)*

`$?x?` in a mapping table entry causes the entry to "succeed" *x* percent of the time; the rest of the time, the entry "fails" and the output of the mapping entry's input is taken unchanged as the output. (Note that, depending upon the mapping, the effect of the entry "failing" is not necessarily the same as the entry not matching in the first place.) *x* should be a real number specifying the success percentage.

For instance, suppose that a system with IP address 123.45.6.78 is sending your site just a little too much e-mail and you'd like to slow it down; if you're using the multithreaded TCP SMTP channel, you can use a `PORT_ACCESS` mapping table in the following way. Suppose you'd like to allow through only 25 percent of its connection attempts and reject the other 75 percent of its connection attempts. The following `PORT_ACCESS` mapping table uses `$?25?` to cause the entry with the `$Y` (accept the connection) to succeed only 25 percent of the time; the other 75 percent of the time, when this entry fails, the initial `$C` on that entry causes IMTA to continue the mapping from the next entry, which causes the connection attempt to be rejected with an SMTP error and the message "Try again later."

```
PORT_ACCESS

TCP|*|25|123.45.6.78|*           $C$?25?$Y
TCP|*|25|123.45.6.78|*           $NTry$ again$ later
```

Sequence Number Substitutions (`$#...#`)

A `$#...#` substitution increments the value stored in an IMTA sequence file and substitutes that value into the template. This can be used to generate unique, increasing strings in cases where it is desirable to have a unique qualifier in the mapping table output; for instance, when using a mapping table to generate file names.

Permitted syntax is any one of the following:

```
$#seq-file-spec|radix|width#
```

```
$#seq-file-spec|radix#
```

```
$#seq-file-spec#
```

The required *seq-file-spec* argument is a full file specification for an already existing IMTA sequence file, and where the optional *radix* and *width* arguments specify the radix (base) in which to output the sequence value, and the number of digits to output, respectively. The default radix is 10. Radices in the range -36 to 36 are also allowed; for instance, base 36 gives values expressed with digits 0,...,9,A,...,Z. By default, the sequence value is printed in its natural width, but if the specified width calls for a greater number of digits, then the output will be padded with 0's on the left to obtain the desired number of digits. Note that if a width is explicitly specified, then the radix must be explicitly specified also.

As noted above, the IMTA sequence file referred to in a mapping must already exist. To create an IMTA sequence file, use the following command:

```
% touch seq-file-spec
```

or

```
% cat >seq-file-spec
```

A sequence number file accessed via a mapping table must be world readable in order to operate properly. You must also have an IMTA user account in order to use such sequence number files.

Mapping Table Substitutions (`$|...|`)

A substitution of the form `$|`*mapping*`,`*argument*`|` is handled specially. IMTA looks for a auxiliary mapping table named *mapping* in the IMTA mapping file, and uses *`argument`* as the input to that named auxiliary mapping table. The named auxiliary mapping table must exist and must set the `$Y` flag in its output if it is successful; if the named auxiliary mapping table does not exist or doesn't set the `$Y` flag, then that auxiliary mapping table substitution fails and the original mapping entry is considered to fail: the original input string will be used as the output string.

Note that when you wish to use processing control metacharacters such as `$C`, `$R`, or `$L` in a mapping table entry that does a mapping table substitution, the processing control metacharacter should be placed to the left of the mapping table substitution in the mapping table template; otherwise the “failure” of a mapping table substitution will mean that the processing control metacharacter will not be seen.

General database substitutions (`${...}`)

A substitution of the form ${*text*} is handled specially. The *text* part is used as a key to access the general database. This database is generated with the IMTA `crdb` utility. If *text* is found in the database, the corresponding template from the database is substituted. If *text* does not match an entry in the database, the input string is used unchanged as the output string.

If a general database exists, it should be world readable to insure that it operates properly.

Note that when wishing to use processing control metacharacters such as $C, $R, or $L in a mapping table entry that does a general database substitution, the processing control metacharacter should be placed to the left of the general database substitution in the mapping table template; otherwise the “failure” of a general database substitution will mean that the processing control metacharacter will not be seen.

Site-supplied Routine Substitutions (`$[...]`)

A substitution of the form `$[`*image*`,`*routine*`,`*argument*`]` is handled specially. The image,routine,argument part is used to find and call a customer-supplied routine. At runtime, IMTA uses `dlopen` and `dlsym` to dynamically load and call the routine *routine* from the shared library *`image`*. The routine *routine* is then called as a function with the following argument list:

```
status = routine (argument, arglength, result, reslength)
```

`argument` and `result` are 252-byte long character string buffers. `argument` and `result` are passed as a pointer to a character string (for example, in C, as `char*`). `arglength` and `reslength` are signed, long integers passed by reference. On input, `argument` contains the *argument* string from the mapping table template, and `arglength` the length of that string. On return, the resultant string should be placed in `result` and its length in `reslength`. This resultant string will then replace the `$[image,routine,argument]` in the mapping table template. The *routine* routine should return 0 if the mapping table substitution should fail and -1 if the mapping table substitution should succeed. If the substitution fails, then normally the original input string will be used unchanged as the output string.

Note that when wishing to use processing control metacharacters such as `$C`, `$R`, or `$L` in a mapping table entry that does a site-supplied routine substitution, the processing control metacharacter should be placed to the left of the site-supplied routine substitution in the mapping table template; otherwise, the "failure" of a mapping table substitution will mean that the processing control metacharacter will not be seen.

The site-supplied routine callout mechanism allows IMTA's mapping process to be extended in all sorts of complex ways. For example, in a `PORT_ACCESS` or `ORIG_SEND_ACCESS` mapping table, a call to some type of load monitoring service could be performed and the result used to decide whether or not to accept a connection or message.

The site-supplied shared library image `image` should be world readable.

Note – This facility is not designed for use by casual users; it is intended to be used to extend IMTA's capabilities system-wide.

Option Files

Global IMTA options, as opposed to channel options, are specified in the IMTA option file.

The IMTA uses an option file to provide a means of overriding the default values of various parameters that apply to the IMTA as a whole. In particular, the option file is used to establish sizes of the various tables into which the configuration and alias files are read.

Locating and Loading the Option File

The option file is the file specified with the `IMTA_OPTION_FILE` option in the IMTA tailor file (`/etc/opt/SUNWmail/imta/imta_tailor`). By default, this is `/etc/opt/SUNWmail/imta/option.dat`.

Option File Format and Available Options

Option files consist of several lines. Each line contains the setting for one option. An option setting has the form:

```
option=value
```

`value` may be either a string or an integer, depending on the option's requirements. If the option accepts an integer value, a base may be specified using notation of the form *b%v*, where b is the base expressed in base 10 and *v* is the actual value expressed in base *b*.

Comments are allowed. Any line that begins with an exclamation point (`!`) is considered to be a comment and is ignored. Blank lines are also ignored in any option file.

The available options are listed in TABLE 3-14.

TABLE 3-14 Option File Options

Options	Description
`ACCESS_ERRORS` (Integer 0 or 1)	IMTA provides facilities to restrict access to channels on the basis of group IDs on the SunOS operating system. If `ACCESS_ERRORS` is set to 0 (the default), when an address causes an access failure IMTA will report it as a "illegal host or domain" error. This is the same error that would occur if the address were simply illegal. Although confusing, this usage nevertheless provides an important element of security in circumstances where information about restricted channels should not be revealed. Setting `ACCESS_ERRORS` to 1 will override this default and provide a more descriptive error.
`ALIAS_HASH_SIZE` (Integer <= 32,767)	This option sets the size of the alias hash table. This in turn is an upper limit on the number of aliases that can be defined in the alias file. The default is 256; the maximum value allowed is 32,767.
`ALIAS_MEMBER_SIZE` (Integer <= 20,000)	This option controls the size of the index table that contains the list of alias translation value pointers. The total number of addresses on the right-hand sides of all the alias definitions in the alias file cannot exceed this value. The default is 320; the maximum allowed is 20,000.
`BLOCK_LIMIT` (Integer > 0)	This option places an absolute limit on the size, in blocks, of any message which may be sent or received with IMTA. Any message exceeding this size will be rejected. By default, IMTA imposes no size limits. Note also that the `blocklimit` channel keyword can be used to impose limits on a per channel basis. The size in bytes of a block is specified with the `BLOCK_SIZE` option.
`BLOCK_SIZE` (Integer > 0)	IMTA uses the concept of a "block" in several ways. For example, the IMTA log files (resulting from placing the `logging` keyword on channels) record message sizes in terms of blocks. Message size limits specified via the `maxblocks` keyword are also in terms of blocks. Normally, an IMTA block is equivalent to 1024 characters. This option can be used to modify this sense of what a block is. Note that the IMTA stores message sizes internally as an integer number of blocks. If the size of a block in bytes is set to a very small value, it is possible for a very large message to cause an integer overflow. A message size of greater than 2**31 blocks would be needed, but this value is not inconceivable if the block size is small enough.

TABLE 3-14 Option File Options

Options	Description
`CHANNEL_TABLE_SIZE` (Integer <= 32,767)	This option controls the size of the channel table. The total number of channels in the configuration file cannot exceed this value. The default is 256; the maximum is 32,767.
`CONVERSION_SIZE` (Integer <= 2000)	This option controls the size of the conversion entry table, and thus the total number of conversion file entries cannot exceed this number. The default is 32.
`DEQUEUE_DEBUG` (0 or 1)	This option specifies whether debugging output from IMTA's dequeue facility (QU) is produced. If enabled with a value of 1, this output will be produced on all channels that use the QU routines. The default value of 0 disables this output.
`DOMAIN_HASH_SIZE` (Integer <= 32,767)	This option controls the size of the domain rewrite rules hash table. Each rewrite rule in the configuration file consumes one slot in this hash table; thus the number of rewrite rules cannot exceed this option's value. The default is 512; the maximum number of rewrite rules allowed is 32,767.
`EXPROUTE_FORWARD` (Integer 0 or 1)	This option controls the application of the `exproute` channel keyword to forward-pointing (To:, Cc:, and Bcc: lines) addresses in the message header. A value of 1 is the default and specifies that `exproute` should affect forward-pointing header addresses. A value of 0 disables the action of the `exproute` keyword on forward-pointing addresses.
`HISTORY_TO_RETURN` (1-200)	The `HISTORY_TO_RETURN` option controls how many delivery attempt history records are included in returned messages. The delivery history provides some indication of how many delivery attempts were made and in some cases indicates the reason the delivery attempts failed. The default value for this option is 20.
`HOST_HASH_SIZE` (Integer <= 32,767)	This option controls the size of the channel hosts hash table. Each channel host specified on a channel definition in the IMTA configuration file (both official hosts and aliases) consumes one slot in this hash table, so the total number of channel hosts cannot exceed the value specified. The default is 512; the maximum value allowed is 32,767.
`ID_DOMAIN` (String)	The `ID_DOMAIN` option specifies the domain name to use when constructing message IDs. By default, the official host name of the local channel is used.

TABLE 3-14 Option File Options

Options	Description
IMPROUTE_FORWARD (Integer 0 or 1)	This option controls the application of the improute channel keyword to forward-pointing (To:, Cc:, and Bcc: lines) addresses in the message header. A value of 1 is the default and specifies that improute should affect forward-pointing header addresses. A value of 0 disables the action of the improute keyword on forward-pointing addresses.
LINE_LIMIT (Integer)	This option places an absolute limit on the overall number of lines in any message that may be sent or received with IMTA. Any message exceeding this limit will be rejected. By default, IMTA imposes no line-count limits. Note also that the linelimit channel keyword can be used to impose limits on a per channel basis.
LINES_TO_RETURN (Integer)	The LINES_TO_RETURN option controls how many lines of message content IMTA includes when bouncing messages. The default is 20.
LOG_CONNECTION (0 or 1)	The LOG_CONNECTION option controls whether connection information—for example, the domain name of the SMTP client sending the message—is saved in the mail.log file. A value of 1 enables connection logging. A value of 0 (the default) disables it.
LOG_FILENAME (0 or 1)	The LOG_FILENAME option controls whether the names of the files in which messages are stored are saved in the mail.log file. A value of 1 enables file name logging. A value of 0 (the default) disables it.
LOG_FORMAT (1, 2, or 3)	The LOG_FORMAT option controls formatting options for the mail.log file. A value of 1 (the default) is the standard format. A value of 2 requests non-null formatting: empty address fields are converted to the string "<>." A value of 3 requests counted formatting: all variable length fields are preceded by "N:," where "N" is a count of the number of characters in the field.

TABLE 3-14 Option File Options

Options	Description

LOG_HEADER (0 or 1)	The LOG_HEADER option controls whether the IMTA writes message headers to the mail.log file. A value of 1 enables message header logging. The specific headers written to the log file are controlled by a site-supplied log_header.opt file. The format of this file is that of other IMTA header option files. For example, a log_header.opt file containing the following would result in writing the first To: and the first From: header per message to the log file. A value of 0 (the default) disables message header logging:
	`To: MAXIMUM=1`
	`From: MAXIMUM=1`
	`Defaults: MAXIMUM=-1`

LOG_LOCAL (0 or 1)	The LOG_LOCAL option controls whether the domain name for the local host is appended to logged addresses that don't already contain a domain name. A value of 1 enables this feature, which is useful when logs from multiple systems running IMTA are concatenated and processed. A value of 0, the default, disables this feature.
LOG_MESSAGE_ID (0 or 1)	The LOG_MESSAGE_ID option controls whether message IDs are saved in the mail.log file. A value of 1 enables message ID logging. A value of 0 (the default) disables it.
LOG_USERNAME (0 or 1)	The LOG_USERNAME option controls whether the user name associated with a process that enqueues mail is saved in the mail.log file. A value of 1 enables user name logging. A value of 0 (the default) disables it.
MAP_NAMES_SIZE (Integer > 0)	The MAP_NAMES_SIZE option specifies the size of the mapping table name table, and thus the total number of mapping table cannot exceed this number. The default is 32.
MAX_ALIAS_LEVELS (Integer)	The MAX_ALIAS_LEVELS option controls the degree of indirection allowed in aliases; that is, how deeply aliases may be nested, with one alias referring to another alias, and so forth. The default value is 10.
MAX_HEADER_BLOCK_USE (Real Number Between 0 and 1)	The MAX_HEADER_BLOCK_USE keyword controls what fraction of the available message blocks can be used by message headers.
MAX_HEADER_LINE_USE (Real Number Between 0 and 1)	The MAX_HEADER_LINE_USE keyword controls what fraction of the available message lines can be used by message headers.

TABLE 3-14 Option File Options

Options	Description
MAX_INTERNAL_BLOCKS (Integer)	The MAX_INTERNAL_BLOCKS option specifies how large (in IMTA blocks) a message IMTA will keep entirely in memory; messages larger than this size will be written to temporary files. The default is 10. For systems with lots of memory, increasing this value may provide a performance improvement.
MAX_LOCAL_RECEIVED_LINES (Integer)	As IMTA processes a message, it scans any Received: header lines attached to the message looking for references to the official local host name. (Any Received: line that IMTA inserts will contain this name.) If the number of Received: lines containing this name exceeds the MAX_LOCAL_RECEIVED_LINES value, the message is entered in the IMTA queue in a held state. The default for this value is 10 if no value is specified in the option file. This check blocks certain kinds of message forwarding loops. The message must be manually moved from the held state for processing to continue.
MAX_RECEIVED_LINES (Integer)	As IMTA processes a message, it counts the number of Received: header lines in the message's header. If the number of Received: lines exceeds the MAX_RECEIVED_LINES value, the message is entered in the IMTA queue in a held state. The default for this value is 50 if no value is specified in the option file. This check blocks certain kinds of message forwarding loops. The message must be manually moved from the held state for processing to continue.
MM_DEBUG (Integer <=5)	The MM_DEBUG option requests various levels of debugging of enqueue routines (handling address rewriting, mapping, conversions, etc.). Both master_debug and slave_debug must be set on a channel in order for MM_DEBUG settings to take affect. The l channel is an exception. Higher settings of MM_DEBUG cause more verbose output.
NORMAL_BLOCK_LIMIT (Integer)	The NORMAL_BLOCK_LIMIT option may be used to instruct IMTA to downgrade the priority of messages based on size: messages above the specified size will be downgraded to non-urgent priority. This priority, in turn, may affect whether the message is processed immediately, or whether it is left to wait for processing until the next periodic job runs.

TABLE 3-14 Option File Options

Options	Description
NON_URGENT_BLOCK_LIMIT (Integer)	The NON_URGENT_BLOCK_LIMIT option may be used to instruct IMTA to downgrade the priority of messages based on size: messages above the specified size will be downgraded to lower than non-urgent priority, meaning that they will not be processed immediately and will wait for processing until the next periodic job runs. The value is interpreted in terms of IMTA blocks, as specified by the BLOCK_SIZE option. Note also that the nonurgentblocklimit channel keyword may be used to impose such downgrade thresholds on a per channel basis.
ORIGINAL_CHANNEL_PROBE (0 or 1)	This option controls whether things like mapping table probes use the original channel as the input channel name, or use the current source channel as the input channel name. The default is 0, meaning to use the current input channel name.
OS_DEBUG (0 or 1)	The OS_DEBUG option requests debugging of OS routines (routines for creating, opening, and closing files and getting system times, etc.). Both master_debug and slave_debug must be set on a channel in order for OS_DEBUG to take affect. l channel is an exception. The output goes to the normal channel debug log files.
POST_DEBUG (0 or 1)	This option specifies whether debugging output is produced by IMTA's periodic delivery job. If enabled with a value of 1, this output will be produced in the post.log file. The default value of 0 disables this output.
RECEIVED_DOMAIN (String)	The RECEIVED_DOMAIN option sets the domain name to use when constructing Received: headers. By default, the official host name of the local channel.
RETURN_ADDRESS (String)	The RETURN_ADDRESS option sets the return address for the local postmaster. The local postmaster's address is postmaster@*localhost* by default, but it can be overridden with the address of your choice. Care should be taken in the selection of this address—an illegal selection may cause rapid message looping and pile-ups of huge numbers of spurious error messages.
RETURN_DEBUG (0 or 1)	The RETURN_DEBUG option enables or disables debugging output in the nightly message bouncer batch job. A value of 0 disables this output (the default), while a value of 1 enables it. Debugging output, if enabled, appears in the output log file, if such a log file is present. The presence of an output log file is controlled by the crontab entry for the return job.

TABLE 3-14 Option File Options

Options	Description
RETURN_DELIVERY_HISTORY (0 or 1)	This flag controls whether or not a history of delivery attempts is included in returned messages. The delivery history provides some indication of how many delivery attempts were made and, in some cases, indicates the reason the delivery attempts failed. A value of 1 enables the inclusion of this information and is the default. A value of 0 disables return of delivery history information. The HISTORY_TO_RETURN option controls how much history information is actually returned.
RETURN_ENVELOPE (Integer)	The RETURN_ENVELOPE option takes a single integer value, which is interpreted as a set of bit flags. Bit 0 (value = 1) controls whether return notifications generated by IMTA are written with a blank envelope address or with the address of the local postmaster. Setting the bit forces the use of the local postmaster address; clearing the bit forces the use of a blank addresses. Note that the use of a blank address is mandated by RFC 1123. However, some systems do not handle blank-envelope-from-address properly and may require the use of this option. Bit 1 (value = 2) controls whether IMTA replaces all blank envelope addresses with the address of the local postmaster. Again, this is used to accommodate noncompliant systems that don't conform to RFC 821, RFC 822, or RFC 1123. Note also that the returnenvelope channel keyword can be used to impose this sort of control on a per channel basis.
RETURN_PERSONAL (String)	The RETURN_PERSONAL option specifies the personal name to use when IMTA generates postmaster messages (for example, bounce messages). By default, IMTA uses the string, "Internet Mail Delivery."
REVERSE_ENVELOPE (0 or 1)	The REVERSE_ENVELOPE option controls whether IMTA applies the address reversal to envelope From: addresses as well as header addresses. This option will have no effect if the USE_REVERSE_DATABASE option is set to 0 or if the reverse database does not exist. The default is 1, which means that IMTA will attempt to apply the database to envelope From: addresses. A value of 0 will disable this use of the address reversal database.
STRING_POOL_SIZE (Integer <= 10,000,000)	The STRING_POOL_SIZE option controls the number of character slots allocated to the string pool used to hold rewrite rule templates and alias list members. A fatal error will occur if the total number of characters consumed by these parts of the configuration and alias files exceeds this limit. The default is 60,000; the maximum allowed value is 10,000,000.

TABLE 3-14 Option File Options

Options	Description
URGENT_BLOCK_LIMIT (Integer)	The URGENT_BLOCK_LIMIT option may be used to instruct IMTA to downgrade the priority of messages based on size: messages above the specified size will be downgraded to normal priority. This priority, in turn, may affect whether the message is processed immediately or left to wait for processing until the next periodic job runs. The value is interpreted in terms of IMTA blocks, as specified by the BLOCK_SIZE option. Note also that the urgentblocklimit channel keyword may be used to impose such downgrade thresholds on a per channel basis.
USE_ALIAS_DATABASE (0 or 1)	The USE_ALIAS_DATABASE option controls whether IMTA makes use of the alias database as a source of system aliases for local addresses. The default is 1, which means that IMTA will check the database if it exists. A value of 0 will disable this use of the alias database.
USE_ERRORS_TO (0 or 1)	The USE_ERRORS_TO option controls whether IMTA makes use of the information contained in Errors-to: header lines when returning messages. Setting this option to 1 directs IMTA to make use of this header line. A value of 0, the default, disable uses of this header line.
USE_REVERSE_DATABASE (0-31)	The USE_REVERSE_DATABASE option controls whether IMTA makes use of the address reversal database and REVERSE mapping as a source of substitution addresses. This value is a decimal integer representing a bit-encoded integer, the interpretation of which is given in TABLE 3-15.
USE_WARNINGS_TO (0 or 1)	The USE_WARNINGS_TO option controls whether IMTA makes use of the information contained in Warnings-to: header lines when returning messages. Setting this option to 1 directs IMTA to make use of these header lines. The default is 0, which disables use of this header line.

TABLE 3-15 USE_REVERSE_DATABASE Bit Values

Bit	Value	Usage
0	1	When set, address reversal is applied to addresses after they have been rewritten by the IMTA address-rewriting process.
1	2	When set, address reversal is applied before addresses have had IMTA address rewriting applied to them.
2	4	When set, address reversal will be applied to all addresses, not just to backward-pointing addresses.

3	8	When set, channel-level granularity is used with REVERSE mapping. REVERSE mapping table (pattern) entries must have the form (note the vertical bars [|])
		
		`source-channel|`
		`destination-channel|`
		`address`

4	16	When set, channel-level granularity is used with address reversal database entries. Reversal database entries must have the form (note the vertical bars [|])
		
		`source-channel|`
		`destination-channel|`
		`address`
		
		Note that Bit 0 is the least significant bit.
		
		The default value for USE_REVERSE_DATABASE is 5, which means that IMTA will reverse envelope From: addresses and both backward- and forward-pointing addresses after they have passed through the normal address-rewriting process. Simple address strings are presented to both REVERSE mapping and the reverse database. Note that a value of 0 disables the use of the address reversal completely.

Header Option Files

Some special option files may be associated with a channel that describe how to trim the headers on messages queued to that channel. This facility is completely general and may be applied to any channel; it is controlled by the `headertrim`, `noheadertrim`, `headerread`, and `noheaderread` channel keywords.

An option file can be used in addition to the channel keywords to configure the behavior of a channel. This configuration tool is available for the Solaris `/var/mail`, the UUCP, the pipe, and the SMTP channels. In addition, any channel can use a header option file in order to create or remove channel-specific headers in messages processed by the channel's master program.

Header option files have a different format than other IMTA option files, and thus a header option file is always a separate file.

Header Option File Location

For header trimming to be applied upon message *dequeue*, IMTA looks in the config directory (`/etc/opt/SUNWmail/imta`) for header options files with names of the form *`channel`*`_headers.opt`, where *`channel`* is the name of the channel with which the header option file is associated. The `headertrim` keyword must be specified on the channel to enable the use of such a header option file.

For header trimming to be applied upon message *enqueue*, IMTA looks in the config directory (`/etc/opt/SUNWmail/imta`) for header options files with names of the form *`channel`*`_read_headers.opt`, where *`channel`* is the name of the channel with which the header option file is associated. The headerread keyword must be specified on the channel to enable the use of such a header option file.

Header option files should be world readable.

Header Option File Format

Simply put, the contents of a header option file are formatted as a set of message header lines. Note, however, that the bodies of the header lines do not conform to RFC 822.

The general structure of a line from a header options file is:

```
Header-name: OPTION=VALUE, OPTION=VALUE, OPTION=VALUE, ...
```

`Header-name` is the name of a header line that IMTA recognizes (any of the header lines described in this manual may be specified, plus any of the header lines standardized in RFC 822, RFC 987, RFC 1049, RFC 1421, RFC 1422, RFC 1423, RFC 1424, RFC 1327, and RFC 1521 (MIME).

Header lines not recognized by IMTA are controlled by the special header line name Other:. A set of options to be applied to all header lines not named in the header option file can also be given on a special Defaults: line. Use of Defaults: guards against the inevitable expansion of IMTA's known header line table in future releases.

Various options may then be specified to control the retention of the corresponding header lines. The available options listed in TABLE 3-16.

TABLE 3-16 Header Option File Options

Option	Description
`ADD` (Quoted String)	The `ADD` option creates a completely new header line of the given type. The new header line contains the specified string. The header line created by `ADD` will appear after any existing header lines of the same type. The `ADD` option cannot be used in conjunction with the header line type; it will be ignored if it is specified as part of an Other: option list.
`FILL` (Quoted String)	The `FILL` option creates a completely new header line of the given type if and only if there are no existing header lines of the same type. The new header line contains the specified string The `FILL` option cannot be used in conjunction with the header line type; it will be ignored if it is specified as part of an Other: option list.
`GROUP` (Integer 0 or 1)	This option controls grouping of header lines of the same type at a particular precedence level. A `GROUP` value of 0 is the default, and indicates that all header lines of a particular type should appear together. A value of 1 indicates that only one header line of the respective type should be output and the scan over all header lines at the associated level should resume, leaving any header lines of the same type unprocessed. Once the scan is complete it is then repeated in order to pick up any remaining header lines. This header option is primarily intended to accommodate Privacy Enhanced Mail (PEM) header processing.
`MAXCHARS` (Integer)	This option controls the maximum number of characters which may appear in a single header line of the specified type. Any header line exceeding that length is truncated to a length of `MAXCHARS`. This option pays no attention to the syntax of the header line and should never be applied to header lines containing addresses and other sorts of structured information. The length of structured header lines should be controlled with the `maxheaderchars` and `maxheaderaddrs` channel keywords.
`MAXIMUM` (Integer)	This option controls the maximum number of header lines of this type that may appear. This has no effect on the number of lines; after wrapping, each individual header line might consume. A value of -1 is interpreted as a request to suppress this header line type completely.

TABLE 3-16 Header Option File Options

Option	Description
MAXLINES (Integer)	This option controls the maximum number of lines all header lines of a given type may occupy. It complements the MAXIMUM option in that it pays no attention to how many header lines are involved, only to how many lines of text they collectively occupy. As with the MAXIMUM option, headers are trimmed from the bottom to meet the specified requirement.
PRECEDENCE (Integer)	This option controls the order in which header lines are output. All header lines have a default precedence of zero. The smaller the value, the higher the precedence. Thus, positive PRECEDENCE values will push header lines toward the bottom of the header while negative values will push them toward the top. Equal precedence ties are broken using IMTA's internal rules for header line output ordering.

Job Controller

The job controller is responsible for scheduling and executing the message delivery or message submission tasks upon request by various IMTA components. For example, upon receipt of an incoming message from any source, the IMTA channel that is handling the receipt of the message determines the destination, enqueues the message, and sends a request to the job controller to execute the next channel. The job controller schedules only the client tasks for IMTA.

Internally, the job controller maintains the set of channel queues. Requests are placed on specified queues by server processes as messages are processed. Each queue has a job limit that consists of the maximum number of concurrent jobs that can be processed and the maximum number of jobs that can be enqueued. Requests are executed as they are received until the job limit is exceeded, at which point they are queued to run when a currently executing request finishes. If the capacity of a queue is exceeded, requests directed at that queue are ignored by the job controller.

Job Controller Configuration

At startup, the job controller reads a configuration file that specifies parameters, queues, and channel processing information. This configuration information is specified in the file `job_controller.cnf` in the `/etc/opt/SUNWmail/imta/` directory.

The job controller configuration file, `job_controller.cnf`:

- Defines various types of queues that differ by their capacity and job limit
- Specifies for all channels the master program name and the slave program name, if applicable

In the `imta.cnf` file, you can specify a type of queue (that was defined in `job_controller.cnf`) by using the queue *keyword*. For example, the following fragment from a sample `job_controller.cnf` file defines the queue `MY_QUEUE`:

```
[QUEUE=MY_QUEUE]
capacity = 300
job_limit = 12
```

The following fragment from a sample `imta.cnf` file specifies the queue `MY_QUEUE` in a channel block:

```
channel_x queue MY_QUEUE
channel_x-daemon
```

If you want to modify the parameters associated with the default queue configuration or add additional queues, you can do so by editing the `job_controller.cnf` file, and stopping and then restarting the job controller with the command:

```
# imta restart job_controller
```

A new job controller process is created, using the new configuration, and receives subsequent requests. The old job controller process continues to execute any requests it has queued until they are all finished, at which time it exits.

To stop the job controller, execute the following command:

```
# imta stop job_controller
```

The first queue in the job controller configuration file, by default the only queue, is used for any requests that do not specify the name of a queue. IMTA channels defined in the IMTA configuration file (`imta.cnf`) may have their processing requests directed to a specific queue by using the queue channel keyword followed by the name of the queue. The queue name must match the name of a queue in the job controller configuration. If the job controller does not recognize the requested queue name, the request are ignored.

Examples of Use

Typically, you would add additional types of queue characteristics to the job controller configuration if you wanted to differentiate processing of some channels from that of other channels. You might also choose to use queues with different characteristics. For example, you might need to control the number of simultaneous requests that some channels are allowed to process. You can do this by creating a new queue with the desired job limit, then use the queue channel keyword to direct those channels to the new, more appropriate queue.

In addition to the definition of queues, the job controller configuration file also contains a table of IMTA channels and the commands that the job controller must use to process requests for each channel. These two types of requests are termed “master” and “slave.” Typically, a channel master program is invoked when there is a message stored in an IMTA message queue for the channel. The master program dequeues the message and delivers it.

A slave program is invoked to poll a channel and pick up any messages inbound on that channel. While nearly all IMTA channels have a master program, many do not need a slave program. For example, a channel that handles SMTP over TCP/IP doesn’t use a slave program because a network service, the SMTP server, receives incoming SMTP messages upon request by any SMTP server. The SMTP channel's master program is IMTA’s SMTP client.

A `master_shutdown` command may be associated with each channel that contains master programs. This will be the command to stop the master program if the job controller is stopped. Such commands are useful for master programs which run like daemons. The format is:

```
master_shutdown = path
```

path is the full pathname to the shutdown executable.

If the destination system associated with the channel cannot handle more than one message at a time, you need to create a new type of queue whose job limit is one:

```
[QUEUE = single_job]
job_limit = 1
capacity = 200
```

On the other hand, if the destination system has enough parallelism, you can set the job limit to a higher value. The capacity defines the maximum number of requests which the job_controller will store at given time. Requests that are received after the limit has been reached are ignored.

Job Controller Configuration File Format

In accordance with the format of IMTA option files, the job controller configuration file contains lines of the form:

```
option=value
```

In addition to option settings, the file may contain a line consisting of a section and value enclosed in square-brackets ([ ]) in the form:

```
[section-type=value]
```

Such a line indicates that option settings following this line apply only to the section named by value. Initial option settings that appear before any such section tags apply globally to all sections. Per section option settings override global defaults for that section. Recognized section types for the job controller configuration file are `QUEUE,` to define queues and their parameters, and `CHANNEL,` to define channel processing information.

The following is a sample job controller configuration file (`job_controller.cnf`).

```
!IMTA job controller configuration file
!
!Global defaults
debug=1
udp_port=27442(1)
args=""
slave_command=NULL(2)
capacity=100(3)
!
!
!Queue definitions
!
[QUEUE=DEFAULT](4)
job_limit=10(5)
capacity=200
!
[QUEUE=SINGLE_JOB]
job_limit=1
capacity=200
!
!
!Channel definitions
!
!
[CHANNEL=l](6)
master_command=/opt/SUNWmail/imta/lib/l_master
!
[CHANNEL=sims-ms]
master_command=/opt/SUNWmail/ims/lib/ims_master
!
[CHANNEL=tcp_*](7)
master_command=/opt/SUNWmail/imta/lib/tcp_smtp_client
```

The key items in the preceding example (numbered, enclosed in parentheses, and in bold font) are:

1. This global option defines the UDP port number on which the job controller listens for requests.
2. Sets a default `SLAVE_COMMAND` for subsequent [`CHANNEL`] sections.
3. Sets a default `CAPACITY` for subsequent [`QUEUE`] sections.
4. This [`QUEUE`] section defines a queue named `DEFAULT`. Since this is the first queue in the configuration file, it is used by all channels that do not specify a queue name using the queue channel keyword.

5. Set the JOB_LIMIT for this queue to 10.
6. This [CHANNEL] section applies to a channel named l, the IMTA local channel. The only definition required in this section is the master_command, which the job controller issues to run this channel. Since no wild card appears in the channel name, the channel must match exactly.
7. This [CHANNEL] section applies to any channel whose name begins with tcp_*. Since this channel name includes a wild card, it will match any channel whose name begins with tcp_.

The available options are:

CAPACITY=*integer*	Specifies the maximum number of outstanding requests that a queue can hold. Additional requests beyond the CAPACITY of the queue are ignored. Exceeding the CAPACITY of a queue does not affect the ability of another queue to buffer outstanding requests until that queue's CAPACITY is exceeded. If set outside of a section, it is used as the default by any [QUEUE] section that doesn't specify CAPACITY. This option is ignored inside of a [CHANNEL] section.
DEBUG=0 or 1	If DEBUG=1 is selected, IMTA writes debugging information to a file in the /var/opt/SUNWmail/imta/log directory named job_controller-*uniqueid*, where *uniqueid* is a unique ID string that distinctively identifies the file name. The purge utility recognizes the *uniqueids* and can be used to remove older log files.)
JOB_LIMIT=*integer*	Specifies the maximum number of requests that a queue can execute in parallel. Execution of a request uses a UNIX system process, so this corresponds to the maximum number of UNIX system processes you allow a queue to use. If more requests are present for a queue, they are held until an executing job finishes, unless the CAPACITY of the queue is exceeded. The JOB_LIMIT applies to each queue individually; the maximum total number of jobs is the sum of the JOB_LIMIT parameters for all queues. If set outside of a section, it is used as the default by any [QUEUE] section that doesn't specify JOB_LIMIT. This option is ignored inside of a [CHANNEL] section.

MASTER_COMMAND=*file specification*	Specifies the full path to the command to be executed by the UNIX system process created by the job controller in order to run the channel and dequeue messages outbound on that channel. If set outside of a section, it is used as the default by any [CHANNEL] section that doesn't specify a MASTER_COMMAND. This option is ignored inside of a [QUEUE] section.
SLAVE_COMMAND=*file specification*	Specifies the full path to the command to be executed by the UNIX system process created by the job controller in order to run the channel and poll for any messages inbound on the channel. Many IMTA channels do not have a SLAVE_COMMAND. If that is the case, the reserved value NULL should be specified. If set outside of a section, it is used as the default by any [CHANNEL] section that doesn't specify a SLAVE_COMMAND. This option is ignored inside of a [QUEUE] section.
UDP_PORT=*integer*	Specifies the UDP port on which the job controller should listen for request packets. Do not change this option unless the default conflicts with another UDP application on your system. If you do change this option, be sure to change the corresponding IMTA_JBC_SERVICE option in the IMTA tailor file, /etc/opt/SUNWmail/imta/imta_tailor, so that it matches. The UDP_PORT option applies globally and is ignored if it appears in a [CHANNEL] or [QUEUE] section.

A master_shutdown command may be associated with each channel that contains master programs. This will be the command to stop the master program if the job controller is stopped. Such commands are useful for master programs which run like daemons. The format is:

```
master_shutdown = path
```

path is the full pathname to the shutdown executable.

SMTP Dispatcher

The IMTA multithreaded SMTP Dispatcher is a multithreaded connection dispatching agent that permits multiple multithreaded servers to share responsibility for a given service. When using the SMTP Dispatcher, it is possible to have several multithreaded SMTP servers running concurrently. In addition to having multiple servers for a single service, each server may handle simultaneously one or more active connections.

Note – The SMTP Dispatcher is available only in the Sun Internet Mail Server 3.5 - Enterprise Edition.

Operation of the SMTP Dispatcher

The SMTP Dispatcher works by acting as a central receiver for the TCP ports listed in its configuration. For each defined service, the IMTA SMTP Dispatcher may create one or more SMTP server processes that actually handle the connections after they've been established.

In general, when the SMTP Dispatcher receives a connection for a defined TCP port, it checks its pool of available SMTP server processes and chooses the best candidate for the new connection. If no suitable candidate is available and the configuration permits it, the SMTP Dispatcher creates a new SMTP server process to handle this and subsequent connections. The SMTP Dispatcher may also proactively create a new SMTP server process in expectation of future incoming connections. There are several configuration options that can tune the IMTA SMTP Dispatcher's control of its various services, and in particular, to control the number of SMTP server processes and the number of connections each SMTP server process handles.

Creation and Expiration of SMTP Server Processes

There are automatic housekeeping facilities within the SMTP Dispatcher to control the creation of new and expiration of old or idle SMTP server processes. The basic options that control the SMTP Dispatcher's behavior in this respect are `MIN_PROCS` and `MAX_PROCS`. `MIN_PROCS` provides a guaranteed level of service by having a number of SMTP server processes ready and waiting for incoming connections. `MAX_PROCS`, on the other hand, sets an upper limit on how many SMTP server processes may be concurrently active for the given service.

Because it is possible that a currently running SMTP server process might not be able to accept any connections either because it is already handling the maximum number of connections of which it is capable, or because the process has been scheduled for termination, the SMTP Dispatcher may create additional processes to assist with future connections.

The `MIN_CONNS` and `MAX_CONNS` options provide a mechanism to help you distribute the connections among your SMTP server processes. `MIN_CONNS` specifies the number of connections that flags a SMTP server process as “busy enough” while `MAX_CONNS` specifies the “busiest” that a SMTP server process can be.

In general, the SMTP Dispatcher creates a new SMTP server process when the current number of SMTP server processes is less than `MIN_PROCS` or when all existing SMTP server processes are “busy enough” (the number of currently active connections each has is at least `MIN_CONNS` and at least 75 percent of `MAX_CONNS`).

If a SMTP server process is killed unexpectedly, for example, by the UNIX system `kill` command, the SMTP Dispatcher still creates new SMTP server processes as new connections come in.

SMTP Dispatcher Configuration File

The SMTP Dispatcher configuration information is specified in the `/etc/opt/SUNWmail/imta/dispatcher.cnf` file. A default configuration file is created at installation time and can be used without any changes made. However, if you want to modify the default configuration file for security or performance reasons, you can do so by editing the `dispatcher.cnf` file.

Note – The `dispatcher.cnf` file is available only in the Sun Internet Mail Server 3.5 - Enterprise Edition.

Configuration File Format

The SMTP Dispatcher configuration file format is similar to the format of other IMTA configuration files. Lines specifying options have the following form:

```
option=value
```

option is the name of an option and *value* is the string or integer to which the options is set. If the *option* accepts an integer *value*, a base may be specified using notation of the form *b*%*v*, where *b* is the base expressed in base 10 and *v* is the actual value expressed in base *b*. Such option specifications are grouped into sections corresponding to the service to which the following option settings apply, via lines of the following form:

```
SERVICE=service-name
```

service-name is the name of a service. Initial option specifications that appear before any such section tag apply globally to all sections.

The following is a sample SMTP Dispatcher configuration file (`dispatcher.cnf`).

```
! The first set of options, listed without a [SERVICE=xxx]
! header, are the default options that will be applied to all
! services.
!
MIN_PROCS=0
MAX_PROCS=5
MIN_CONNS=5
MAX_CONNS=20
MAX_LIFE_TIME=86400
MAX_LIFE_CONNS=100
MAX_SHUTDOWN=2
!
! Define the services available to Dispatcher
!
[SERVICE=SMTP]
PORT=25
IMAGE=/opt/SUNWmail/imta/bin/tcp_smtp_server
LOGFILE=/var/opt/SUNWmail/imta/log/tcp_smtp_server.log
```

The available options are:

`HISTORICAL_TIME=`*integer*	Controls how long the expired connections (those that have been closed) and processes (those that have exited) remain listed for statistical purposes.
`IMAGE=`*file specification*	Specifies the image that is run by SMTP server processes when created by the SMTP Dispatcher. The specified image should be one designed to be controlled by the SMTP Dispatcher.
`LOGFILE=`*file specification*	Causes the SMTP Dispatcher to direct output for corresponding SMTP server processes to the specified file.
`MAX_CONNS=`*integer*	Affects the SMTP Dispatcher's management of connections. This value specifies a maximum number of connections that may be active on any SMTP server process.
`MAX_IDLE_TIME=`*integer*	Specifies the maximum idle time for a SMTP server process. When a SMTP server process has had no active connections for this period, becomes eligible for shutdown. This option is only effective if there are more than the value of `MIN_PROCS` SMTP server processes currently in the SMTP Dispatcher’s pool for this service.

`MAX_LIFE_CONNS`	Specifies the maximum number of connections a SMTP server process can handle in its lifetime. Its purpose is to perform worker-process housekeeping.
`DEBUG`	0 or 1.
`MAX_LIFE_TIME=`*integer*	Requests that SMTP server processes be kept only for the specified number of seconds. This is part of the SMTP Dispatcher's ability to perform worker-process housekeeping. When a SMTP server process is created, a countdown timer is set to the specified number of seconds. When the countdown time has expired, the SMTP server process is subject to shutdown.
`MAX_PROCS=`*integer*	Controls the maximum number of SMTP server processes that are created for this service.
`MAX_SHUTDOWN=`*integer*	Specifies the maximum number of SMTP server processes available before the SMTP Dispatcher shuts down. In order to provide a minimum availability for the service, the SMTP Dispatcher does not shut down SMTP server processes that might otherwise be eligible for shutdown if shutting them down results in having fewer than `MAX_SHUTDOWN` SMTP server processes for the service. This means that processes that are eligible for shutdown can continue running until a shutdown "slot" is available.
`MIN_CONNS=`*integer*	Determines the minimum number of connections that each SMTP server process must have before considering the addition of a new SMTP server process to the pool of currently available SMTP server processes. The SMTP Dispatcher attempts to distribute connections evenly across this pool.
`MIN_PROCS=`*integer*	Determines the minimum number of SMTP server processes that are created by the SMTP Dispatcher for the current service. Upon initialization, the SMTP Dispatcher creates this many detached processes to start its pool. When a process is shut down, the SMTP Dispatcher ensures that there are at least this many available processes in the pool for this service.
`PORT=`*integer1, integer2,...*	Specifies the TCP port(s) to which the SMTP Dispatcher listens for incoming connections for the current service. Connections made to this port are transferred to one of the SMTP server processes created for this service. Specifying `PORT=0` disables the current service.

`STACKSIZE`	Specifies the thread stack size of the SMTP server. The purpose of this option is to reduce the chances of the SMTP server running out of stack when processing deeply nested MIME messages (several hundreds of levels of nesting). Note that these messages are in all likelihood spam messages destined to break mail handlers. Having the SMTP server fail will protect other mail handlers further down the road.

Controlling the SMTP Dispatcher

The SMTP Dispatcher is a single resident process that starts and shuts down SMTP server processes for various services, as needed. The SMTP Dispatcher process is started using the command:

```
# imta start dispatcher
```

This command subsumes and makes obsolete any other `imta start` command that was used previously to start up a component of IMTA that the SMTP Dispatcher has been configured to manage. Specifically, you should no longer use `imta start smtp`. An attempt to execute any of the obsoleted commands causes IMTA to issue a warning.

To shut down the SMTP Dispatcher, execute the command:

```
# imta stop dispatcher
```

What happens with the SMTP server processes when the SMTP Dispatcher is shut down depends upon the underlying TCP/IP package. If you modify your IMTA configuration or options that apply to the SMTP Dispatcher, you must restart the SMTP Dispatcher so that the new configuration or options take effect. To restart the SMTP Dispatcher, execute the command:

```
# imta restart dispatcher
```

Restarting the SMTP Dispatcher has the effect of shutting down the currently running SMTP Dispatcher, then immediately starting a new one.

Tailor File

The IMTA tailor file is an option file in which the location of various IMTA components are set. Certain parameters for tuning the performance of the IMTA databases are also set in this file. This file must always exist in the `/etc/opt/SUNWmail/imta` directory for the MTA to function properly. The file may be edited to reflect the changes in a particular installation. Some options in the file should not be edited. The IMTA should be restarted after making any changes to the file. It is preferable to make the changes while the MTA is down. If the database tuning parameters are changed, all existing IMTA databases must be removed and reconstructed.

Note – It is not recommended that this file be edited unless absolutely necessary.

An option setting has the form:

```
option=value
```

value may be either a string or an integer, depending on the option's requirements. Comments are allowed. Any line that begins with an exclamation point is considered to be a comment and is ignored. Blank lines are also ignored.

Options that are available and can be edited are:

`IMTA_ADMIN_PROPERTY`	Location of the adminserver properties file. The `imta dirsync` utility reads this file to find the domains the IMTA is responsible for. The default value is `/etc/opt/SUNWmail/admin/adminserver.prope rties`.
`IMTA_ALIAS_DATABASE`	The IMTA's alias database. The default is `/var/opt/SUNWmail/imta/db/aliasesdb`.
`IMTA_ALIAS_FILE`	The IMTA's aliases file. Aliases not set in the directory, for example. postmaster, are set in this file. The default value is `/etc/opt/SUNWmail/imta/aliases`.
`IMTA_AUTHDL_DATABASE` (Enterprise Edition only)	Database used for storing distribution lists access controls. The default is `/var/opt/SUNWmail/imta/authdldb`.
`IMTA_CHARSET_DATA`	Specifies where IMTA's compiled character set data is located. The default value is `/opt/SUNWmail/imta/lib/charset_data`.

`IMTA_CHARSET_OPTION_FILE`	File used for charset conversion options. The default is `/etc/opt/SUNWmail/imta/option_charset.dat`.
`IMTA_COM`	Specifies where IMTA's shell scripts are located. The default value is `/opt/SUNWmail/imta/lib/`.
`IMTA_CONFIG_DATA`	The compiled configuration for the IMTA. The default value is `/opt/SUNWmail/imta/lib/config_data`.
`IMTA_CONFIG_FILE`	The IMTA’s configuration file. All it’s rewrite rules and per channel options are set in this file. The default is `/etc/opt/SUNWmail/imta/imta.cnf`.
`IMTA_CONVERSION_FILE`	File to set rules for the conversion channel. The default value is `/etc/opt/SUNWmail/imta/conversions`.
`IMTA_DB_HASH_SIZE`	The IMTA database hash size. The default is 7901. All IMTA databases should be removed and reconstructed if this value is changed.
`IMTA_DB_PTR_SIZE`	The IMTA database pointer size. This value should be increased for very large databases.All IMTA databases should be removed and reconstructed if this value is changed The default value is 10 and works fine for databases with up to 4 million entries.
`IMTA_DISPATCHER_CONFIG` (Enterprise edition only)	The IMTA dispatcher’s configuration file. The default is `/etc/opt/SUNWmail/imta/dispatcher.cnf`.
`IMTA_DOMAIN_DATABASE` (Enterprise Edition only)	This option is not used for SIMS 3.5.
`IMTA_FORWARD_DATABASE` (Enterprise Edition only)	This option is not used for SIMS 3.5.
`IMTA_GENERAL_DATABASE` (Enterprise Edition only)	This database is provided for each site’s customer usage. Generally lookups can be embedded in mappings and rewrite rules. The default is `/var/opt/SUNWmail/imta/generaldb`.
`IMTA_JBC_CONFIG_FILE`	The IMTA job_controller’s configuration file. The default is `/etc/opt/SUNWmail/imta/job_controller.cnf`.
`IMTA_JBC_SERVICE`	Specifies the host and port for the job_controller. *Do not edit this option.*
`IMTA_LANG`	Locale of the IMTA’s notary messages. By default it is `/etc/opt/SUNWmail/imta/locale/C/LC_MESSAGES`.

`IMTA_LDAP_SERVER`	This option specifies the location of the LDAP directory, searched by the IMTA dirsync, autoreply and other programs. The list consists of one or more ldaphost: port pairs separated by commas. Each program reads this list and connects to the first directory, it is able to connect to. It connects to port 389, if the port is not specified. The default is just localhostname:389.
`IMTA_LIB`	Directory where the IMTA libraries and executables are stored. The default is `/opt/SUNWmail/imta/lib/`.
`IMTA_LIBUTIL`	The IMTA's utility library. By default it is `/opt/SUNWmail/lib/libimtautil.so.1`.
`IMTA_LOG`	Location of the IMTA's log files. The default is `/var/opt/SUNWmail/imta/log/`.
`IMTA_MAPPING_FILE` (Enterprise edition only)	File used for setting access control rules, reverse mapping rules, forward mapping rules etc. The default value is `/etc/opt/SUNWmail/imta/mappings`.
`IMTA_NAME_CONTENT_FILE`	Location of file used by the IMTA for content-type conversions. The default is `/etc/opt/SUNWmail/imta/name_content.dat`.
`IMTA_OPTION_FILE`	Name of the IMTA's option file. The default is `/etc/opt/SUNWmail/imta/option.dat`.
`IMTA_QUEUE`	The IMTA's message queue directory. The default is `/var/opt/SUNWmail/imta/queue`.
`IMTA_QUEUE_CACHE_DATABASE`	Location of the IMTA's message queue cache. The default is `/var/opt/SUNWmail/imta/queue_cache/`.
`IMTA_RETURN_PERIOD`	Controls the return of expired messages and the generation of warnings.The default value for this option is 1. If this options is set to an integer value N, then the associated action will only be performed every N times the return job runs. By default the return job runs once every day.
`IMTA_RETURN_SPLIT_PERIOD`	Controls splitting of the mail.log file. The default value for this option is 1. If this options is set to an integer value N, then the associated action will only be performed every N times the return job runs. By default the return job runs once every day.
`IMTA_RETURN_SYNCH_PERIOD`	Controls queue synchronization.The default value for this option is 1. If this options is set to an integer value N, then the associated action will only be performed every N times the return job runs. By default the return job runs once every day.

`IMTA_REVERSE_DATABASE` (Enterprise Edition only)	The IMTA's reverse database. This database is used for rewriting From: addresses. The default is `/var/opt/SUNWmail/imta/db/reversedb`.
`IMTA_ROOT`	Base directory for the IMTA installation. The default is `/opt/SUNWmail/imta/`.
`IMTA_SCRATCH`	Directory where the IMTA stores it's backup configuration files. During a full dirsync temporary database files are also created under this directory.The default is `/var/opt/SUNWmail/imta/tmp/`.
`IMTA_SYNCH_CACHE_PERIOD`	Controls the queue synchronization by the post program.The default value for this option is 1. If this options is set to an integer value N, then the associated action will only be performed every N times the post job runs. By default the post job runs once every four hours.
`IMTA_TABLE`	The IMTA's configuration directory. The default is `/etc/opt/SUNWmail/imta/`.
`IMTA_USER`	Name of the postmaster. The default is inetmail. If this is changed be sure to edit `/etc/opt/SUNWmail/imta/aliases` file to reflect the change to the postmaster address.
`IMTA_USER_PROFILE_DATABASE`	Database used for storing user's vacation, forwarding and program delivery information. The default is `/var/opt/SUNWmail/imta/profiledb`.
`IMTA_USER_USERNAME`	This option specifies the userid of the subsidiary account the IMTA uses for certain "non-privileged" operations—operations which it doesn't want to perform under the usual IMTA account. The default is nobody.
`IMTA_VERSION_LIMIT`	Maximum versions of log files to be preserved while purging old log files. The default value is 5.
`IMTA_VERSION_LIMIT_PERIOD`	Controls the frequency of purging of log files by the post job. The default value for this option is 1. If this options is set to an integer value N, then the associated action will only be performed every N times the post job runs. By default the post job runs once every four hours
`IMTA_WORLD_GROUP`	This is the group in which one must belong to be able to perform certain "privileged" sorts of operations. The default is mail.

Security Considerations

This section discusses various email security issues and techniques to use to provide a secure and effective system.

Controlling Email Access

You can control which users can send messages to which users, what channels can send messages to what channels, impose limits on the size of messages allowed through, and use hooks in the IMTA to allow for dynamic, load-based rejection decisions.

Statically Controlling E-mail Access

In the mapping file (`/etc/opt/SUNWmail/imta/mappings`) the `PORT_ACCESS` mapping table can be used to control from what IP numbers IMTA servers will accept connection attempts; the IMTA multithreaded SMTP server checks this table when a connection attempt comes in. The `ORIG_SEND_ACCESS` mapping table can be used to control, based on From address, To address, and source and destination channel, what messages the IMTA allows to pass through. `ORIG_MAIL_ACCESS` can be used to control email access based on both IP addresses, and From address and To addresses. It essentially combines the functionality of `PORT_ACCESS` and `ORIG_SEND_ACCESS`.

Note – `ORIG_SEND_ACCESS` replaced `SEND_ACCESS` in SIMS 3.2.

SMTP Connection Control Mapping

The multi-threaded SMTP server is able to selectively accept or reject incoming SMTP connections based on IP address and port number. At SMTP server startup time, the server process will look for a mapping table named `PORT_ACCESS`. If present, the server will format connection information in the form:

```
TCP|server-address|server-port|client-address|client-port
```

The server will try to match against all PORT_ACCESS mapping entries. If the result of the mapping contains $N, the connection will be immediately closed. Any other result of the mapping indicates that the connection is to be accepted. $N may optionally be followed by a rejection message. If present, the message will be sent back down the connection just prior to closure. Note that a CRLF terminator will be appended to the string before it is sent back down the connection.

For example, the following mapping will only accept SMTP connections from a single network, except for a particular host singled out for rejection without any message:

```
PORT_ACCESS

  TCP|*|*|192.123.10.70|*    $N
  TCP|*|*|192.123.10.*|*     $Y
  TCP|*|*|*|*                $N500$ Bzzzzzzzzt$ thank$ you$ for$ playing.
```

Note that if you are using the PORT_ACCESS mapping table you will need to restart the IMTA after making any changes to this mapping table so that the IMTA will see the new compiled configuration.

The PORT_ACCESS mapping table is specifically intended for performing only IP number based rejections; for more general control at the email address level or a combination of email address and port access, the ORIG_SEND_ACCESS mapping table or the ORIG_MAIL_ACCESS table can be used respectively.

SMTP Envelope Access Control Mapping

The ORIG_SEND_ACCESS mapping table may be used to control who may or may not send mail, receive mail, or both. The nature of the mapping is very general and allows per channel granularity.

If the ORIG_SEND_ACCESS mapping table exists, then for each recipient of every message passing through the IMTA, the IMTA will probe the table with a probe string of the form (note the use of the vertical bar character, |):

```
src-channel|from-address|dst-channel|to-address
```

src-channel is the channel originating the message (i.e., queueing the message); *from-address* is the address of the message's originator; *dst-channel* is the channel to which the message will be queued; and *to-address* is the address to which the message is addressed. Use of an asterisk in any of these four fields causes that field to match any channel or address, as appropriate.

Now, if the probe string matches a pattern (i.e., the left hand side of an entry in the table), then the resulting output of the mapping is checked. If the output contains the metacharacters `$Y` then the enqueue for that particular `To:` address is permitted. If the mapping output contains the metacharacters `$N` then the enqueue to that particular address is rejected. In the case of a rejection, an optional rejection message may be supplied in the mapping output. This string will be returned as a rejection message. If no string is output (other than the `$N` metacharacter), then a default message will be used.

Suppose that local users in the domain `acme.com`, with the exception of the postmaster, are not allowed to send mail to the Internet but can receive mail from there. Then the `ORIG_SEND_ACCESS` mapping table shown in CODE EXAMPLE 3-1 is one possible way to enforce this restriction. In that example, the local host name is assumed to be `acme.com`. In the channel name `tcp_*`, wild cards are used so as to match any possible TCP/IP channel name. In the rejection message, dollar signs are used to quote spaces in the message. Without those dollar signs, the rejection would be ended prematurely and only read “Internet” instead of “Internet postings are not permitted”.

CODE EXAMPLE 3-1 Restricting Internet Mail Access on UNIX

```
ORIG_SEND_ACCESS

  *|postmaster@acme.com|*|*  $Y
  *|*|*|postmaster@acme.com  $Y
  l|*@acme.com|tcp_*|*      $NInternet$ postings$ are$ not$
permitted
```

Mail Access Mapping

The third and last access control mapping table combines the two previous mapping tables: `ORIG_SEND_ACCESS` and `PORT_ACCESS`. The syntax is similar (it should appear on one line):

```
TCP|server-address|server-port|client-address|client-port|SMTP|MAIL|src-channel|
from-address|dest-channel|to-address
```

For example, to allow `abc.com` to relay mail through your domain only if the mail is submitted from the IP address 192.9.9.9, use the following mapping:

```
ORIG_MAIL_ACCESS

  TCP|*|*|192.9.9.9|*|SMTP|MAIL|tcp_local|*@abc.com|tcp_local|*    $Y
  TCP|*|*|*|*|SMTP|MAIL|tcp_local|*@abc.com|tcp_local|*            $N
```

ORIG_SEND_ACCESS is looked up first. If ORIG_SEND_ACCESS has a rule which supersedes the rule in ORIG_MAIL_ACCESS, the rules in ORIG_MAIL_ACCESS will not be used at all.

Dynamically Controlling Email Access

When implementing dynamic rejection mechanisms, the TCP/IP channel options ALLOW_TRANSACTIONS_PER_SESSION and ALLOW_RECIPIENTS_PER_TRANSACTION may be of interest. The ALLOW_TRANSACTIONS_PER_SESSION option can be used to limit the number of messages accepted during a particular connection. After refusing a number of connection attempts from a particular site, once you do let them connect, they are liable to have a backlog of messages for your site which they will try to deliver during that connection. If you are attempting to "slow down" how much mail you accept from that site, you likely will want to use this option to say, in effect, "enough for now" after some point in the connection. Similarly, the ALLOW_RECIPIENTS_PER_TRANSACTION option can be used to limit the number of recipients allowed for a particular message; this can be useful in protecting against a denial of service attack in the form of messages blanketing large numbers of your users.

Imposing Message Size Limits

The IMTA options BLOCK_LIMIT and LINE_LIMIT can be used to impose global size limits on all IMTA channels. The channel keywords blocklimit and linelimit can be used to impose size limits on specific channels.

Restricting or Controlling Information Emitted

This section describes various ways information that you may not wish to emit can leak out and describes ways of blocking this.

SMTP Probe Commands

During an SMTP connection, a remote sending side (or a person manually telnetting to your SMTP port) can issue commands requesting information such as a check on the validity of addresses. This very useful information can, however, be subject to abuse, for example, by automated search engines checking for valid email addresses on your firewall system. Therefore some sites may have an interest in disabling these helpful features.

Setting DISABLE_EXPAND=1 in your Internet TCP/IP channel option file disables the SMTP EXPN command. The SMTP EXPN command is normally used to expand (get the membership of) mailing lists.

Setting HIDE_VERIFY=1 in your Internet TCP/IP channel option file causes the IMTA to return a "generic" response to the SMTP VRFY command. The SMTP VRFY command is normally used to check whether an address is a legitimate address on the local system. (Note that as it is required that SMTP servers support the VRFY command, the IMTA has to return some sort of response; with HIDE_VERIFY=1, this response is simply a "maybe" sort of response rather than an explicit yes or no.)

Setting DISABLE_ADDRESS=1 in your Internet TCP/IP channel option file causes the IMTA to disable responses to the IMTA SMTP server's private XADR command, which normally returns information about the channel an address matches.

Setting DISABLE_STATUS=1 in your Internet TCP/IP channel option file causes the IMTA to disable responses to the IMTA SMTP server's private XSTA command, which normally returns information about the numbers of messages in IMTA queues.

Setting DISABLE_GENERAL=1 in your Internet TCP/IP channel option file causes the IMTA to disable responses to the IMTA SMTP server's private XGEN command, which normally returns status information about whether an IMTA compiled configuration and character set are in use.

A sample TCP/IP channel option file to disable probing via the SMTP server, for a site using a tcp_local channel, would be as shown in the following example.

```
DISABLE_EXPAND=1
HIDE_VERIFY=1
DISABLE_ADDRESS=1
DISABLE_STATUS=1
DISABLE_GENERAL=1
```

Internal names in `Received:` headers

`Received:` headers are normally exceptionally useful headers for displaying the routing that a message really took. Their worth can be particularly apparent in cases of dealing with apparently forged email, or in cases where one is trying to track down what happened to a broken messages, or in cases where a message does not appear to be replyable and one is trying to figure out who might know how to respond to the message. `Received:` headers are also used by the IMTA and other mailers to try to detect message loops.

`Message-id:` headers are normally useful for message tracking and correlation.

However, on the converse side, `Received:` headers on messages you send out give the message recipient information about the routing that a message really took through your internal systems and tend to include internal system names and possibly an envelope recipient address. And `Message-id:` headers tend to include internal system names. At some sites, this may be considered a security exposure.

If your site is concerned about this information being emitted, first see if you can configure your internal systems to control what information they put in these headers. For instance, the IMTA options `RECEIVED_DOMAIN` and `ID_DOMAIN` can be used on an IMTA system to specify the domain name to use when constructing Received: headers and `Message-id:` headers, respectively. Although these options are not usually particularly relevant on the IMTA firewall system itself—after all, the firewall system is by definition a system whose name is intended to be visible to the outside world—if you have the IMTA on internal systems also, the options may be of interest on those internal IMTA systems. In a similar spirit, the channel keyword noreceivedfor can be used on channels on an IMTA system to instruct the IMTA not to include the envelope recipient address in the `Received:` header it constructs, if limiting the exposure of internal "routing" addresses is a concern for your site. Only if you cannot configure your internal systems to control such sorts of information should you consider resorting to stripping such headers off entirely.

`Received:` and `Message-id:` headers should not be removed lightly, due to their many and important uses, but if the internal routing and system name information in them is sensitive for your site and if you cannot configure your internal systems to control what information appears in these headers, then you may wish to strip off those headers on messages going out to the Internet via header trimming on your outgoing TCP/IP channel.

Note – Do not remove `Received:` or `Message-id:` headers on general principles or because your users do not like them. Removing such headers, among other things, (1) removes one of the best tracking mechanisms you have, (2) removes information that may be critical in tracking down and solving problems, (3) removes one of the few (and best) warnings of forged mail you may have, and (4) blocks the mail system's ability to detect and short-circuit message loops. Only remove such headers if you know your site needs them removed.

To implement header trimming, put the `headertrim` keyword—you will probably want the `innertrim` keyword as well—on your outgoing external TCP/IP channel or channels, generally `tcp_local` and possibly other `tcp_*` channels (possibly every `tcp_*` channel except your internal channel, `tcp_internal`), where the x depends upon the TCP/IP package you are using, and create a header trimming file for each such channel. The headertrim keyword causes header trimming to be applied to the outer message headers; the `innertrim` keyword causes the header trimming to be applied also to embedded message parts (MESSAGE/RFC822 parts) within the message. A sample header trimming file for a site using a `tcp_local` channel is shown in the following example.

```
Received: MAXIMUM=-1
MR-Received: MAXIMUM=-1
X400-Received: MAXIMUM=-1
Message-id: MAXIMUM=-1
```

Logging and Tracking Messages

This section points out some message logging and tracking techniques.

Identifying the Source of Incoming SMTP Messages

The `identtcp` or `identtcpnumeric` channel keyword when placed on your *x*`tcp_local` channel, where *x* depends on just which sort of TCP/IP channel you're using, causes the IMTA to attempt an `IDENT` query on incoming SMTP connections. If the sending system is running an `IDENT` server, it will return to the IMTA the SMTP sender's identity for the IMTA to insert in the Received: header the IMTA constructs. If the sending system is not running an `IDENT` server, the IMTA will just use the port number (port 25) and the sending system IP number or name.

With `identtcpnumeric`, the IMTA uses the `IDENT` information (if any) and the actual IP number of the sending system; with `identtcp`, the IMTA also attempts to translate the IP number to a system name by performing a DNS reverse lookup. Thus `identtcpnumeric` incurs slightly less overhead because it does not do the DNS reverse lookup, and the actual IP number may perhaps be considered somewhat more authoritative that the name resulting from a DNS query. However, using the system name as with `identtcp` may be considered more user-friendly.

Identifying information in `Received:` headers can assist in detecting spoofed e-mail and in holding the senders of such spoofed e-mail accountable. Note that user-friendly identifying information is a not insignificant feature: even a naive user may notice that a `Received:` header in a suspicious message contains an unexpected address, for example, `anonymous@SpoofersAreUs.edu`, but only a fairly sophisticated user is liable to pay attention to any IP numbers showing up in `Received:` headers. So a choice between these keywords may be affected by whether you are looking to provide forewarning to users that they may have received spoofed e-mail, or whether you merely wish to preserve the identifying information for use in investigating cases of spoofed e-mail.

Logging Messages Passing through the IMTA

The IMTA provides facilities for logging each message as it is enqueued and dequeued. All log entries are made to the `mail.log_current` file in the IMTA log directory, `/var/opt/SUNWmail/imta/log/mail.log_current`. Logging is controlled on a per-channel basis. The logging keyword activates logging for a particular channel while the nologging keyword disables it. Logging is enabled on all channels by default.

When logging is turned on, the cumulative `mail.log` file in the IMTA log directory will continue to grow and grow; the IMTA itself never does anything with this log file and it is up to you to periodically write it to backup and delete it, or truncate it, or whatever your site prefers.

The log file is written as normal ASCII text and the format is quite simple. By default, each entry contains eight or nine fields. For example:

```
19-Jan-1998 19:16:57.64 l tcp_default_router D 1 adam@acme.com marlow@alpha.com
smtp:250<marlowe@alpha.com> Recipient ok
```

The fields are described in TABLE 3-17:

TABLE 3-17 Log File Fields

`19-Jan-1998 19:16:57.64`	The date and time when the entry was made.
`l` (lower case "L")	The channel name for the source channel.
`tcp_default_router`	The channel name of the destination channel. For SMTP channels when `LOG_CONNECTION` is enabled, a minus sign (`-`) indicates inbound to the SMTP server, a plus sign (`+`) indicates outbound via the SMTP client.
`D`	The type of entry. See TABLE 3-18.
`1`	The size of the message. This is expressed in kilobytes by default, although this default can be changed by using the `BLOCK_SIZE` keyword in the IMTA option file.

TABLE 3-17 Log File Fields

`adam@acme.com`	The envelope `From:` address. Note that for messages with an empty envelope `From:` address, such as notification messages, this field will be blank.
`marlow@alpha.com`	The original form of the envelope `To:` address.
`smtp:250<marlow@alpha.com>`	The active (current form of the envelope `To:` address. (SMTP channels during dequeue only).
`Recipient ok`	The delivery status (SMTP channels only during dequeue).

The logging entry codes are described in TABLE 3-18:

TABLE 3-18 Logging Entry Codes

Entry	Description
General	
D	Successful dequeue
E	Enqueue
J	Access control mapping of attempted enqueue
Q	Temporary failure to dequeue
R	Recipient address rejected on attempted dequeue
Z	Some successful recipients, but this recipient was temporarily unsuccessful; the original message file of all recipients was dequeued, and in its place a new message file for this and other unsuccessful recipients will be immediately re-enqueued.
SMTP Channels' LOG_CONNECTION + or - entries	
C	Connection closed
O	Connection opened
X	Connection rejected
Y	Connection try failed before being established

In addition to the base set of data logged when the `logging` keyword is used, there are options to cause the log output to include additional details.

Extra Logging Detail

In addition to the base set of logging enabled via the logging channel keyword, the IMTA has options that cause additional information to be included in the entries written to the `mail.log*` files. Note that logging such additional information tends to incur additional overhead.

In particular, setting `LOG_MESSAGE_ID=1`, `LOG_CONNECTION=1`, and `LOG_FILENAME=1` in your IMTA option file may be of interest on an IMTA e-mail firewall. Logging the message ID makes it easier to find entries in the log file corresponding to a particular message, or to correlate different entries in the log file corresponding to a single message. Logging the SMTP client connection information can be useful to show just what system really sent the message to your IMTA firewall. Logging the filename can be useful if you wish to correlate log file entries with actual message files currently in the IMTA queue area.

Setting `LOG_HEADER=1` may be of interest if you wish to save certain message headers to the `mail.log*` files.

Additionally, setting `LOG_PROCESS=1` and `LOG_USERNAME=1` on an IMTA firewall system ought generally to result in fairly monotonous extra information being logged: the process id of the process enqueuing a message on an IMTA firewall system would normally be that of an IMTA SMTP server process (for SMTP messages), and the username would normally just be the username of the user who last restarted the IMTA Service Dispatcher. Enable these options if you wish to confirm that the process ids and usernames of processes enqueuing messages are as expected.

Note – It is up to the individual sites whether or not to implement a log cleaning policy. By default, `mail.log` is never removed; this can potentially fill up your disk space.

Snapshots of Message Traffic through the IMTA

The IMTA maintains channel counters based on the Mail Monitoring MIB, RFC 1566. These counters can provide "snapshots" of the state of the IMTA queues and a feel for the volume of messages passing through the IMTA.

Dirsync Option File

This file is used to set options for the `dirsync` program that cannot be set through the command line. This file should be located in the IMTA's configuration directory, which is specified by the value for `IMTA_TABLE` in the `imta_tailor` file. In this file, any line that begins with an exclamation point is considered to be a comment and is ignored. Blank lines are also ignored. The format of this file is:

```
option=value
```

value may be either a string or an integer, depending on the option's requirements.

If any of the options in this file are changed, perform a full `dirsync` after the change.

The available options are:

`IMTA_DL_DIR`	Directory where the distribution lists member's list files are stored. Default value is `/var/opt/SUNWmail/imta/dl/`.
`IMTA_DL_HASHSIZE`	Maximum number of subdirectories under the `dl` directory. This number must be a prime number. Default value is `211`.
`IMTA_PROGRAM_CONFIG`	File where information about delivery programs are stored. The default is `/etc/opt/SUNWmail/imta/program.opt`.
`IMTA_PROGRAM_DIR`	Location of the programs used for program delivery. The default is `/opt/SUNWmail/imta/programs/`.

Autoreply Option File

This file is used for setting options for the autoreply or vacation program. This file should be located in the IMTA's configuration directory, which is specified by the value for `IMTA_TABLE` in the `imta_tailor` file. In this file, any line that begins with an exclamation point is considered to be a comment and is ignored. Blank lines are also ignored The format of this file is:

```
option=value
```

value may be either a string or an integer, depending on the option's requirements.

The available options available:

`DEBUG`	This option determines whether a trace file is created for each autoreply. By default the value is 0 and this facility is off. A value of 1, creates an autoreply trace file for each autoreply sent in the IMTA's log directory. A value of 3 puts more information in the trace file.
`RESEND_TIMEOUT`	If mail arrives for a recipient with autoreply on, an autoreply is not sent if a certain period has not elapsed since the last autoreply was sent from this recipient to this specific sender. This option sets the time in hours, after which an autoreply is sent to the same sender again. The default value, if this option is not set, is 168 (for example, once a week).

Aliases, Reverse Mapping, and Forward Mapping

The IMTA provides a facility to support mailbox names associated with the local system that do not necessarily correspond to actual users: *aliases*. Aliases are useful for constructing mailing lists, forwarding mail, and synonyms for user names. A second set of related facilities provides support for "centralized naming," whereby you establish, for instance, mail addresses of the form `first.last@acme.com` for all of your users. There are several advantages to such centralized naming systems: the addresses are simple, they provide added security in that they make no reference to internal account or system names, and, because they lack reference to account and system names, they are more stable.

Each time an address that matches the local channel is encountered by the IMTA's message submission logic, the mailbox (for example, username) specified in the address is compared against each entry in the alias database or alias file. If a match occurs the alias address is replaced by the translation value or values specified by the alias. An alias can translate into any combination and number of additional aliases or real addresses. The real addresses need not themselves be associated with the local channel and thus aliases can be used to forward mail to remote systems.

Aliases only apply to addresses mapped to the local channel; furthermore, note that since the only addresses truly considered to match a channel are Envelope To: addresses, aliases can only apply to Envelope To: addresses. The IMTA performs alias translation and expansion only after address parsing is completed. The translation values produced by an alias are treated as completely new addresses and are reprocessed from scratch.

Alias File

This file is used to set aliases not set in the directory. In particular, the postmaster alias is a good example. Aliases set in this file will be ignored, if the same aliases exist in the directory. The IMTA has to be restarted for any changes to take effect. Any line that begins with an exclamation point is considered to be a comment and is ignored. Blank lines are also ignored.

A physical line in this file is limited to 252 characters. You can split a logical line into multiple physical lines using the backslash (\) continuation character.

The format of the file is as follows:

```
<user>: <address>
```

For example:

```
! A /var/mail/ user
inetmail: inetmail@mailhost

! A message store user
ms_testuser: mstestuser@sims-ms.mailhost
```

Including Other Files in the Alias File

Other files can be included in the primary alias file. A line of the following form directs the IMTA to read the `file-spec` file:

```
<file-spec
```

The file specification must be a complete file path specification and the file must have the same protections as the primary alias file; for example, it must be world readable.

The contents of the included file are inserted into the alias file at its point of reference. The same effect can be achieved by replacing the reference to the included file with the file's actual contents. The format of include files is identical to that of the primary alias file itself. Indeed, include files may themselves include other files. Up to three levels of include file nesting are allowed.

The Address-Reversal Database and REVERSE Mapping

Header From: addresses and other backward-pointing headers receive one additional processing step. This processing can be extended to all header addresses if the third bit (bit 2) in the IMTA option USE_REVERSE_DATABASE is set. While `uid@mailhost.alpha.com` is the fully qualified form of the address, mail recipients, especially those outside the company, should not see the address in that form. The reverse database allows you to specify a per-user return address, which is the `preferredRfc822Originator` address in the directory.

The reverse database is created each time you run the `imta dirsync` command.

The address-reversal database is generally located in the IMTA database directory. The database is the files whose names are specified with the IMTA_REVERSE_DATABASE option in the `/etc/opt/SUNWmail/imta/imta_tailor` file, which by default are the files `/var/opt/SUNWmail/imta/db/reversedb.*`.

Note – Do not edit this database directly. Any required changes must be done in the directory.

If the address is found in the database, the corresponding right-hand side from the database is substituted for the address. If the address is not found, an attempt is made to locate a mapping table named REVERSE in the mapping file. No substitution is made and rewriting terminates normally if the table does not exist or no entries from the table match.

Reverse mapping can also be performed on a per-channel basis. `src_channel|` destination and `channel|` internal addresses need to be mapped to `*|tcp_local|*@*.acme.com` and `$|@acme.com$Y`.

If the address matches a mapping entry, the result of the mapping is tested. The resulting string will replace the address if the entry specifies a $Y; a $N will discard the result of the mapping. If the mapping entry specifies $D in addition to $Y, the resulting string will be run through the reversal database once more; and if a match occurs, the template from the database will replace the mapping result (and hence the address).

As an example, suppose that the internal addresses at acme.com are actually of the form user@host.acme.com, but, unfortunately, the user name space is such that user@hosta.acme.com and user@hostb.acme.com specify the same person for all hosts at acme.com. Then the following, very simple REVERSE mapping may be used in conjunction with the address-reversal database:

```
REVERSE
*@*.acme.com            $0@host.acme.com$Y$D
```

This mapping maps addresses of the form user@host.acme.com to user@host.acme.com. The $D metacharacter causes the address-reversal database to be consulted. The address-reversal database should contain entries of the form:

```
user@host.acme.com     first.last@acme.com
```

The reverse and noreverse channel keywords, and the IMTA options USE_REVERSE_DATABASE and REVERSE_ENVELOPE may used to control the specifics of when and how address reversal is applied. In particular, address reversal will not be applied to addresses in messages when the destination channel is marked with the noreverse keyword. If USE_REVERSE_DATABASE is set to 0, address reversal will not be used with any channel. The REVERSE_ENVELOPE option controls whether or not address reversal is applied to envelope From: addresses as well as message header addresses. See the descriptions of these options and keywords for additional information on their effects. By default, the address reversal database is used if the routability scope is set to the mail server domains.

FORWARD Address Mapping

Address reversals are not applied to envelope To: addresses. The reasons for this omission are fairly obvious—envelope To: addresses are continuously rewritten and modified as messages proceed through the mail system. The entire goal of routing is to convert envelope To: addresses to increasingly system- and mailbox-specific formats. The canonization functions of address reversal are entirely inappropriate for envelope To: addresses.

The various substitution mechanisms for envelope To: addresses provide functionality equivalent to the reversal database, but none of these things provides functionality equivalent to reverse mapping. And circumstances do arise where mapping functionality for envelope To: addresses is useful and desirable.

The FORWARD mapping table provides this missing functionality. If a FORWARD mapping table exists in the mapping file, it is applied to each envelope To: address. No changes are made if this mapping does not exist or no entries in the mapping match.

If the address matches a mapping entry, the result of the mapping is tested. The resulting string will replace the envelope To: address if the entry specifies a $Y; a $N will discard the result of the mapping.

The following example illustrates the use of a complex REVERSE and FORWARD mapping. Suppose that a system or pseudo domain named am.sigurd.nocompany.com associated with the l channel produces RFC 822 addresses of the general form:

```
"lastname, firstname"@am.sigurd.nocompany.com
```

or

```
"lastname,firstname"@am.sigurd.nocompany.com
```

Although these addresses are perfectly legal, they often confuse other mailers that do not fully comply with RFC 822 syntax rules—mailers that do not handle quoted addresses properly, for instance. Consequently, an address format that does not require quoting tends to operate with more mailers. One such format is:

```
firstname.lastname@am.sigurd.nocompany.com
```

The goals of this example mapping are to:

- Allow any of these three address formats to be used
- Present only addresses in the original format to the mr_local channel, converting formats as necessary
- Present only addresses in the new unquoted format to all other channels, converting formats as necessary.

The following mapping file tables produce the desired results. The REVERSE mapping shown assumes that bit 3 in the IMTA option USE_REVERSE_DATABASE is set.

```
REVERSE
   *|mr_local|"*,$ *"@am.sigurd.nocompany.com $Y"$1,$ $2"@am.sigurd.nocompany.com
   *|mr_local|"*,*"@am.sigurd.nocompany.com   $Y"$1,$ $2"@am.sigurd.nocompany.com
   *|*|"*,$ *"@am.sigurd.nocompany.com        $Y$3.$2@am.sigurd.nocompany.com
   *|*|"*,*"@am.sigurd.nocompany.com          $Y$3.$2@am.sigurd.nocompany.com
   *|mr_local|*.*@am.sigurd.nocompany.com     $Y"$2,$ $1"@am.sigurd.nocompany.com
   *|*|*.*@am.sigurd.nocompany.com            $Y$2.$3@am.sigurd.nocompany.com

FORWARD
   "*,$ *"@am.sigurd.nocompany.com            $Y"$0,$ $1"@am.sigurd.nocompany.com
   "*,*"@am.sigurd.nocompany.com              $Y"$0,$ $1"@am.sigurd.nocompany.com
   *.*@am.sigurd.nocompany.com                $Y"$1,$ $0"@am.sigurd.nocompany.com
```

CHAPTER **4**

Sun Directory Services Configuration

The Sun Directory Services stores naming and addressing information for the components of SIMS, and stores profiles for the users of those components. For example:

- The MTA uses the directory to store e-mail user profiles and distribution lists.
- The Legacy Services Hub uses the directory to synchronize information held in various proprietary directories and messaging systems.
- The Internet Message Access server uses the directory to verify the credentials of users and to store user message store configuration information.

Configuration decisions for the directory service must be made for the whole service, and not in isolation for each component or each server. Before you configure a component of the directory service, make sure you understand the overall consequences for other components of the choices you make.

This section describes the default configuration for the Sun Directory Services. It also explains how to perform the following tasks:

- Configure general and default properties
- Create a data store
- View and modify the schema
- Create and modify access control rules
- Configure logging parameters

Initial Configuration Properties

When you install the directory service software, most configurable characteristics are given default settings that enable you to start and run a directory server. The only items that you *must* configure are:

- Name of the administrator
- Password for the administrator (the default is `secret`)
- Distinguished Name (DN) of the naming context held in the data store by the server
- Location of the data store

Note – Although command-line configuration activities are described in detail in this chapter, the recommended procedures for configuration are most easily accomplished using the System Administrator Console Graphical User Interface (GUI), described in the *Sun Internet Mail Server 3.5 Administrator's Guide*.

Specifying the Administrator Name and Password

The system administrator name and password are stored in the configuration file, so that the administrator always has access to the directory. This is necessary for the administrator to solve problems with access control, for example. You do not need to be authenticated as the administrator to perform directory maintenance. Any user who has `root` access can update directory information.

For additional information, refer to Chapter 2, "Commands Reference," in the sections entitled "`imta cnbuild`", and "`imldifsync`."

Password Management

A directory entry for a SIMS user contains a `userPassword` attribute. The value of this attribute, which is used to authenticate the user to the directory, can be stored in encrypted or unencrypted format. See "Default Configuration" for details of how to specify whether or not passwords are stored in an encrypted format. By default, passwords are encrypted.

When you supply a password for authentication, or as an attribute value in a directory operation, you specify the value in unencrypted format. You do not have to enter the password in its encrypted format.

The directory service software uses the `crypt(1)` utility to encrypt passwords.

Specifying the Data Store Naming Context and Location

You must specify the distinguished name of the naming context held in the data store on the server and the directory where the data store files are held. For a default configuration, you do not need to specify any other information. (For more information about creating a data store, see Chapter 2, “Commands Reference” in the section entitled “Message Access and Store.”)

Default Configuration

When you have specified the information described in “Initial Configuration Properties”, you have a server with a default configuration. It has the following characteristics:

- Port used for Lightweight Directory Access Protocol (LDAP) communications is 389
- Port used by the LDAP/HTTP gateway is 1760
- Searches are limited to 5000 entries or 3600 seconds (one hour). A search stops when the first of these limits is reached
- Schema is checked for each directory operation.
- Data store is in `/var/opt/SUNWconn/ldap/dbm`
- Log files are stored in `/var/opt/SUNWconn/ldap/log`
- 1000 entries are cached. A cache file of 100,000 bytes is created for each data store.
- Default indexing is used, as described in “`idxgen`”
- Passwords are stored in encrypted format
- Alias dereferencing on bind operation is enabled
- Directory contains no entries (the name and password for the administrator are stored in the configuration file)
- Default access control is used, as described in the table located in the following section “Configuring Access Control.”
- No knowledge references apply to other servers

IMTA is distributed with a default job controller configuration that is suitable for most sites. This default configuration defines a single queue named DEFAULT with a job limit of 4 and a capacity of 200. As the first queue in the file, DEFAULT is used by all IMTA channels that do not specify a queue using the queue channel keyword. (In the default configuration, the queue DEFAULT is actually the only queue.) In addition, the supplied job controller configuration file includes channel definitions for all of the supplied and supported IMTA channels.

The job controller configuration file is required. If it is not present or its contents are incorrect, the job controller does not start. There is no need to modify the configuration file unless you choose to add queues, modify queue parameters, or add processing information for locally developed channels.

LDAP Directory Concepts

This section provides information on:

- Directory Information (database content)
- Access Control
- Directory Structure
- Replication

Directory Information

The information held in the directory consists of entries and alias entries, which is the information base, and of infrastructure information, which determines how the information base is structured.

Directory Entries

A *directory entry* is a set of *attributes* and their *values*. Every entry has an *object class* attribute, which specifies the kind of object the entry describes, and defines the set of attributes it contains. Some attributes are mandatory and some are optional. The schema defines the attributes that are mandatory and optional for an entry of a given object class.

Directory information is hierarchical, with entries organized in a tree structure. Each entry has a parent entry and can have child entries. The top of the hierarchy is known as the *root entry*.

An entry is identified by its *distinguished name* (DN). A distinguished name is a sequence of attributes and values. The first attribute is the *naming attribute* of the entry. This attribute and its value provide the entry's *relative distinguished name* (RDN). The rest of the sequence is the distinguished name of the parent entry. A distinguished name is unique throughout the whole directory service.

FIGURE 4-1 shows an example of how directory information is structured, with the DNs and RDNs of the shaded entries.

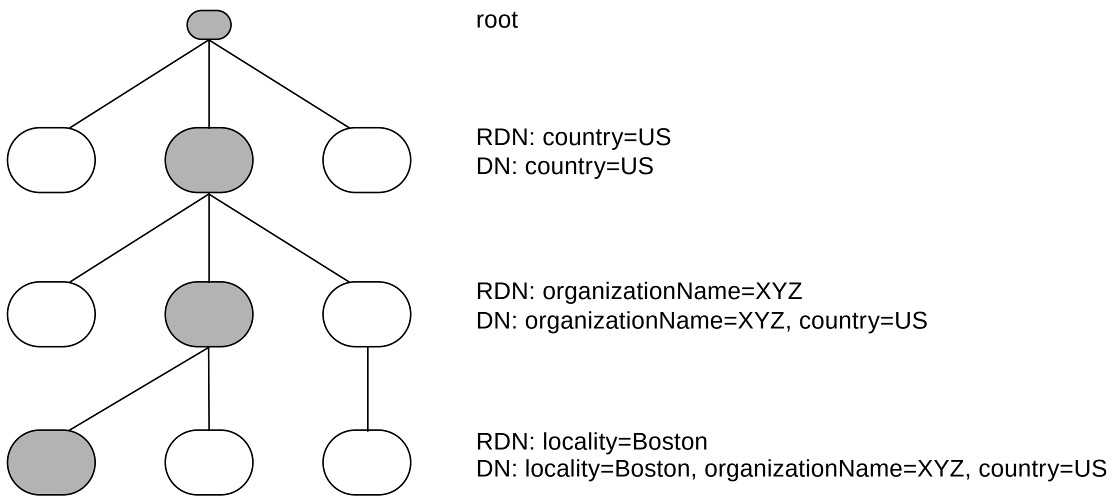
root
RDN: country=US
DN: country=US
RDN: organizationName=XYZ
DN: organizationName=XYZ, country=US
RDN: locality=Boston
DN: locality=Boston, organizationName=XYZ, country=US

FIGURE 4-1 Directory Information Structure

The directory information is divided into naming contexts. A *naming context* is a subtree of the directory, and is identified by the DN of the entry at the top of the naming context. A naming context is stored in a *data store*. A data store can hold more than one naming context.

In a general-purpose directory, you have to decide what information you want to store, and how that information will be organized. The SIMS directory service has already been designed for you, though you can modify this design, as described in "Modifying the Schema."

Aliasing

You can define an *alias entry*. An alias entry is identified by a distinguished name. It contains the name of the directory entry it represents (the aliased object name) and the naming attribute of the alias entry. The alias entry and the entry it represents must be in the same data store.

For bind and search operations, you can specify that the directory should translate an alias DN to the DN of the actual entry. This is known as *dereferencing* the alias. For other operations, you need to treat the alias entry as an ordinary entry and not dereference it, for example, to modify the RDN of the alias entry itself, not of the aliased object.

Alias Entries and Searching

The result of a search or read operation involving an alias entry differs depending on whether or not you dereference the alias. For example, suppose your directory contains the following pair of entries:

```
cn=Stan Smith, role=Personnel Administrator, ou=Personnel,
ou=Corporate, o=XYZ, c=US
with attributes:  objectclass=orgPerson
                  cn=Stan Smith
                  telephoneNumber=123 456 7890
                  mail=dtmail
cn=personnel, o=XYZ, c=US
with attributes:  objectclass=alias
                  cn=personnel
                  aliasedObjectName=”cn=Stan Smith, role=Personnel
                  Administrator, ou=Personnel, ou=Corporate, o=XYZ,
                  c=US”
```

There are four possible settings for the alias `dereference` flag:

- Never dereference alias–All operations apply to the entry with the given DN, even though the entry is an alias entry. This is the default setting.
- Dereference alias when finding base object–The base object identifies the top of the subtree of entries to be searched. This setting means that if you specify an alias as the base object it will be dereferenced, but no other aliases encountered during the search are dereferenced.

- Dereference alias when searching–If the operation being carried out is a search, all alias entries specified or used in the search are dereferenced. If the result of the search is an alias entry, the aliased object is returned to the user, not the alias entry. This can sometime lead to unexpected results to searches based on DN content, where the requested information is not present in the entries returned, because the entry that contains the requested DN term is an alias entry that has been dereferenced.
- Always dereference alias–All alias entries specified or used in the operation are dereferenced.

For example, with alias dereferencing when searching, if you search for the telephone number of `cn=Personnel Administrator`, `o=XYZ`, `c=US`, you will see Stan Smith's telephone number. With no alias dereferencing, you would not see a telephone number.

Defining aliases for roles is particularly useful when the person occupying a role changes frequently (the duty network manager for out-of-hours calls, for example), so that users always query the same entry. You can change the value of the `aliasedObjectName` with a script that runs on a schedule and calls `ldapmodify` to make the changes.

See "`ldapsearch`" for details of how to specify how alias dereferencing is used in `ldapsearch`.

Alias Entries and Authentication

Every interaction with the directory starts with a bind request, to authenticate the user and establish the level of access permitted. The DN supplied in a bind request can be the DN of an alias entry. With alias dereferencing, the user binds with the DN contained in the aliasedObjectName of the alias entry, and is granted the access rights defined for the entry with that DN.

If aliases are not being dereferenced, the user binds with the DN of the alias entry and is refused access because the password attribute is not present.

Infrastructure Information

Infrastructure information determines how the components of a directory service behave and how directory entry information is interpreted. It includes the directory schema, knowledge information, and component configuration information.

Directory Service Configuration Files

The directory service uses a number of configuration files to control the activity of its components:

- `slapd.conf` contains `slapd` configuration information. It includes the object class and attribute configuration information, in
  - `slapd.oc.conf`
  - `slapd.at.conf`

  See "`slapd.conf` File Format" or the `slapd.conf(4)` man page for a detailed description of the content and format of `slapd.conf`.
- `snmpslapd.conf` contains configuration information used by the SNMP agent monitoring `slapd`. *Do not modify this file.*

The following files contain information used by directory clients:

- `ldapfilter.conf`
- `ldapfriendly.conf`
- `ldapsearchprefs.conf`
- `ldaptemplates.conf`

The following files contain information used by the LDAP/HTTP Gateway.

- `webldapfilter.conf`
- `webldapfriendly.conf`
- `webldaptemplates.conf`

The directory server daemon, `slapd,` reads the current configuration when it is started. If you change the configuration while `slapd` is running, the updated configuration is automatically re-read by the daemon when you save the configuration.

If you make a modification to the configuration, the files in the `current` directory are copied to the `previous` directory. The files are copied only once in the course of an administrative session. This means that you have a copy of the configuration that was in effect before you made any modifications. If you forget to save the changes to the configuration file, the next time you start the Admin Console you are asked whether you want to update the configuration with previously recorded changes.

`slapd.conf` File Format

The file `slapd.conf` contains configuration information for the `slapd`(8) daemon. This configuration information is also used by the `slurpd`(8) replication daemon and by the LDBM indexing utilities `ldif2ldbm(1M)`, `ldif2index(8)`, `ldif2id2entry(8)`, and `ldif2id2children(8)`.

The `slapd.conf` file consists of a series of global configuration options that apply to `slapd` as a whole (including all data stores), followed by zero or more definitions that contain information specific to a data store.

The general format of `slapd.conf` is as follows:

```
# Main Configuration Section
include slapd.at.conf - schema attribute definition file
include slapd.oc.conf - schema object class definition file
global configuration options
# Access Control Section
access control options
# Data Store Section
data store options
...
```

As many data store sections as desired may be included. Global options can be overridden in a data store (for options that appear more than once, the last appearance in the slapd.conf file is used). Blank lines and comment lines beginning with a hash character (#) are ignored. If a line begins with white space, it is considered a continuation of the previous line.

Arguments on configuration lines are separated by white space. If an argument contains white space, the argument must be enclosed in double quotes. If an argument contains a double quote (") or a backslash character (\), the character should be preceded by a backslash character.

The specific configuration options available are described in “Global Configuration Options”, “Access Control Configuration Options”, and “Data Store Options.”

Global Configuration Options

Options described in this section apply to all data stores, unless specifically overridden in a data store definition. Arguments that should be replaced by actual text are *emphasized*.

`attribute` *name* [*name2*] { `bin` | `ces` | `cis` | `tel` | `dn` }	Associate a syntax with an attribute name. By default, an attribute is assumed to have syntax **cis**. An optional alternate name can be given for an attribute. The possible syntaxes and their meanings are: • `bin` - Binary • `ces` - Case-exact string • `cis` - Case ignore string • `tel` - Telephone number string • `dn` - Distinguished name Attribute definitions are stored in `slapd.at.conf` .
`defaultaccess` { `none` | `compare` | `search` | `read` | `write` | `delete` }	Specify the default access to grant requestors not matched by any other access line. The default behavior is to grant read access.
`bindderef` { `on` | `off` }	If `bindderef` is set to `on`, an alias included in a bind request is dereferenced. Otherwise, the alias is used to bind.
`include` *filename*	Read additional configuration information from the given file before continuing with the next line of the current file.
`loglevel` *integer*	Specify the level at which debugging statements and operation statistics should be logged (currently logged to the `syslogd`(8) `LOG_LOCAL4` facility). Log levels are additive, and available levels are: • 1 - Trace function calls • 2 - Debug packet handling • 4 - Heavy trace debugging • 8 - Connection management • 16 - Print out packets sent and received • 32 - Search filter processing • 64 - Configuration file processing • 128 - Access control list processing • 256 - Stats log connections / operations / results • 512 - Stats log entries sent • 2048 - Entry parsing
`objectclass` *name* `requires` *attrs* `allows` *attrs*	Define the schema rules for the object class *name*. These are used in conjunction with the `schemacheck` option. Object class definitions are stored in `slapd.oc.conf`.
`protected` { `crypt` | `none` }	Specifies the method with which passwords stored in the directory are encrypted. A password can be encrypted using `crypt`(1) or not encrypted.

`referral` *url*	Specify the referral to pass back when `slapd`(8) cannot find a local database to handle a request.
`schemacheck` { on | `off` }	Turn schema checking `on` or `off`. The default is `off`.
sizelimit *integer*	Specify the maximum number of entries to return from a search operation. The default size limit is `500`.
timelimit *integer*	Specify the maximum number of seconds (in real time) `slapd` will spend answering a search request. The default time limit is `3600`.

Access Control Configuration Options

The following options specify access control permissions.

`access to` *what* [`by` *who accesslevel*] +	Grant access (specified by *accesslevel*) to a set of entries and / or attributes (specified by *what*) by one or more requestors (specified by *who*).

Data Store Options

Options in this section only apply to the configuration file section for the data store in which they are defined.

`database` *databasetype*	Marks the beginning of a new database instance definition. *databasetype* is always `ldbm`.
`lastmod on` | `off`	Controls whether `slapd` will automatically maintain the `modifiersName`, `modifyTimestamp`, `creatorsName`, and `createTimestamp` attributes for entries. The default value for lastmod is `off`.
`readonly on` | `off`	Puts the database into read-only mode. Any attempts to modify the database will return an "unwilling to perform" error. The default value for readonly is `off`.
`replica host=`*hostname* `[:port`] `"binddn=`*DN*`" bindmethod=simple`	Specifies a replication site for this database.
`replogfile` *filename*	Specifies the name of the replication log file to log changes to. The replication log is typically written by `slapd`(8) and read by `slurpd`(8). See "`slapd.replog` File Format" for more information.

`rootdn` *dn*	Specifies the DN of an entry that is not subject to access control or administrative limit restrictions for operations on this database.
`rootpw` *password*	Specifies a password for the rootdn. The password can be encrypted using `crypt`(1). If the password is encrypted, the value in the file is preceded by {crypt}. If the value is not encrypted, it is preceded by {none}.
`suffix` *dn suffix*	Specifies the DN suffix of queries that will be passed to this data store. Multiple suffix lines can be given and at least one is required for each database definition.
`updatedn` *dn*	This option is only applicable in a replica data store. It specifies the DN allowed to make changes to the replica (typically, this is the DN `slurpd`(8) binds as when making changes to the replica).
`cachesize` *integer*	Specifies the size in entries of the in-memory cache maintained by the data store. The default is 1000 entries.
`dbcachesize` *integer*	Specifies the size in bytes of the in-memory cache associated with each open index file. The default is 100000 bytes.
`directory` *directory*	Specifies the directory where the LDBM files containing the database and associated indexes are located. The default is `/usr/tmp`.
`index` { *attrlist* | `default` } [`pres`, `eq`, `approx`, `sub`, `none`]	Specifies the indexes to maintain for the given attribute. If only an *attrlist* is given, all possible indexes are maintained.
`mode` *integer*	Specifies the file protection mode that newly created database index files should have. The default is `0600`.

Directory Service Log Files

The directory service keeps the following log files:

- `slapd.replog`–Contains records of any change to an entry held by the directory server for which a replica is defined; when `slurpd` runs, it uses the information in the replication log to generate a change request and sends it to the directory server that holds the replica.

Note – Do *not* edit the `slapd.replog` file

- `slapd.log`–Contains information about the activity of the `slapd` daemon

- `slapdtool.log`–Contains information about use of the `slapd` daemon from the Admin Console
- `slurpd.log`–Contains information about the activity of the `slurpd` daemon
- `snmpslapd.log`–Contains information about the activity of the `SNMP` daemon that monitors `slapd`
- `web500gw.log` contains information about the activity of the LDAP/HTTP Gateway.

`slapd.replog` File Format

The file `slapd.replog` is produced by the LDAP daemon, `slapd`, when changes are made to its local database that are to be propagated to one or more replica data stores. The file consists of zero or more records, each one corresponding to a change, addition, or deletion from the database. The file is used by `slurpd`, the LDAP update replication daemon. The records are separated by a blank line. Each record has the following format:

- One or more lines indicating the replicas to which the change is to be propagated:

```
replica: hostname[ :portnumber ]
```

- The time the change took place given, as the number of seconds since 00:00:00 GMT, Jan. 1, 1970, with an optional decimal extension, in order to make times unique. Note that `slapd` does not make times unique, but `slurpd` makes all times unique in its copies of the replication log files.

```
time: integer [ .integer ]
```

- The distinguished name of the entry being changed:

```
dn: distinguishedname
```

- The type of change being made:

```
changetype: [modify|add|delete|modrdn]
```

- The change information, the format of which depends on what kind of change was specified in the preceding. For a `changetype` of modify, the format is one or more of the following:

```
add: attributetype
attributetype: value1
attributetype: value2
...
-
```

For a `replace` modification:

```
replace: attributetype
attributetype: value1
attributetype: value2
...
-
```

For a `delete` modification:

```
delete: attributetype
attributetype: value1
attributetype: value2
...
-
```

If no *attributetype* lines are given, the entire attribute is to be deleted.For a `changetype` of `add`, the format is:

```
attributetype1: value1
attributetype1: value2
...
attributetypeN: value1
attributetypeN: value2
```

For a `changetype` of `modrdn`, the format is:

```
newrdn: newrdn
deleteoldrdn: 0 | 1
```

where a value of 1 for `deleteoldrdn` means to delete the values forming the old rdn from the entry, and a value of 0 means to leave the values as non-distinguished attributes in the entry. For a `changetype` of delete, no additional information is needed in the record.

Access to the `slapd.replog` file is synchronized through the use of `flock`(3) on the file `slapd.replog.lock`. Any process reading or writing this file should obey this locking convention. The format of the values is the LDAP Directory Interchange Format described in “LDAP Data Interchange Format.”

Example

The following sample `slapd.replog` file contains information on one of each type of change.

```
replica: ussales.xyz.co
time: 797612941
dn: cn=Ann Jones, o=XYZ Corporation, c=US
changetype: add
objectclass: person
cn: ann jones
cn: annie jones
sn: jones

replica: ussales.xyz.co
time: 797612973
dn: cn=Ann Jones, o=XYZ Corporation, c=US
changetype: modify
add: description
description: Head of East Coast sales division

replica: ussales.xyz.co
time: 797613020
dn: cn=Ann Jones, o=XYZ Corporation, c=US
changetype: modrdn
newrdn: cn=Ann M Jones
deleteoldrdn: 0
```

LDAP Data Interchange Format

The LDAP Data Interchange Format (LDIF) is used to represent LDAP entries in text form. The `ldif2ldbm`(1M) tools can be used to convert from LDIF format to the LDBM format used by `slapd(8)`. The `ldbmcat(1M)` tool can be used to do the reverse conversion.

The basic form of an LDIF entry is:

```
[id]
dn: distinguished name
attrtype: attrvalue
attrtype: attrvalue
...
```

where *id* is the optional entry ID (a positive decimal number). By default, the database creation tools supply the ID for you. The `ldbmcat`(1M) program, however, produces an LDIF format that includes *id* so that new indexes created are consistent with the existing database. A line may be continued by starting the next line with a single space or tab character, for example,

```
dn: cn=Ann Jones, o=XYZ Corpo
ration, c=US
```

Multiple attribute values are specified on separate lines, for example,

```
cn: Ann Jones
cn: Annie Jones
```

If an *attrvalue* contains a non-printing character, or begins with a space or a colon (:), the *attrtyp* is followed by a double colon and the value is encoded in base 64 notation. For example, the value “begins with a space” would be encoded like this:

`cn:: IGJlZ2lucyB3aXRoIGEgc3BhY2U=`

Multiple entries within the same LDIF file are separated by blank lines.

Example

Here is an example of an LDIF file containing three entries.

```
dn: cn=Ann M Jones, o=XYZ Corporation, c=US
cn: Ann M Jones cn: Ann Jones
objectclass: person
sn: Jones

dn: cn=Frederick Smith, o=XYZ Corporation, c=US
cn: Frederick Smith
objectclass: person
sn: Smith

dn: cn=Elaine Jones, o=XYZ Corporation, c=US
cn: Elaine Jones
objectclass: person
sn: Jones
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
ERXRTc4UG1RV19iZ2hnPk1xeXBkeFxlZ2P/2wBDARESEhgVG
...
```

Notice that the jpegPhoto in Elaine Jones's entry is encoded using base 64.

Configuring General Properties

You can configure the following directory server properties.

- The name and password of the administrator, and whether or not the password is stored encrypted (see “Specifying the Administrator Name and Password”)
- Whether or not the schema is checked when directory information is added or modified
- Whether or not alias entries are dereferenced when a bind request is received
- Default access, that is, the level of access granted to entries and attributes for which access control is not specifically defined
- The ports used by LDAP and the web gateway
- The maximum number of concurrent connections the server can accept
- Search limits (time and number of entries)
- The default directory server for referrals

Configuring the Data Store

To configure a data store, you must specify the distinguished name (DN) of the naming context stored, and the name of the directory where the database files reside. Optional configuration information includes:

- Which attributes are indexed
- Congestion thresholds
- Cache size
- Naming contexts stored
- Naming contexts held in the data store are not replicated to other servers

Indexing

A data store can contain a number of attribute indexes to help optimize the speed of access to directory information. An *attribute index* is a list of entries that contains a given attribute or attribute value. You can index attributes using any of the following matching rules:

- Equality

  Optimizes direct access to entries where an exact attribute value is supplied
- Presence

  Optimizes searches with filters specifying the presence of an attribute but no specific value (`cn=*`, for example)
- Substring

  Optimizes searches with filters specifying a partially specified attribute value (`cn=ad*`, for example)
- Approximate

  Optimizes searches with approximate match filters; an approximate match index is the most expensive in terms of disk space

By default, the following attributes are indexed:

- `commonName`, surname, `mail`, and `mailServer` are indexed by presence, equality, approximate match, and substring match.
- `uid` is indexed by presence and equality

The advantage of indexing is that it optimizes access for indexed attributes. The disadvantages are that it uses more disk space, and that adding and modifying entries takes longer.

When you add or modify an entry after an index has been created, the index is automatically updated. However, if you create a new index and the data store already contains entries, those entries are not automatically included in the index. Indexes are not automatically updated when entries are removed from the directory, so the size of the index files is not reduced as entries are removed. Recreating the indexes for a data store can take several minutes, depending on the number and complexity of the indexes defined. For example, recreating the default indexes for a data store of 20,000 entries takes approximately five minutes.

Congestion Thresholds

Congestion thresholds ensure that the directory does not become overloaded by preventing new operations from starting when there are insufficient resources.

- When the available disk space reaches the *congested* limit, `add` operations are no longer permitted, though `modify`, `modrdn`, `search`, `read`, and `delete` operations are allowed.
- When the database reaches the *critical* limit, only `search`, `read`, and `delete` operations are allowed, and `add`, `modify`, and `modrdn` operations are not permitted.
- The restrictions remain in force until the remaining disk space becomes greater than the *back-to-normal* limit.

The disk that holds the data store tracks the number of free threshold values in Kbytes.

Caching

As information is retrieved from the directory, it is saved in the *cache*. When the cache is full, the oldest entry is discarded to make room for new information. Retrieving information from the cache is faster than retrieving entries from the data base, but a large cache occupies more disk space.

Naming Contexts

By default, a data store contains one naming context. The distinguished name (DN) of this naming context is used to identify the data store. However, a data store can contain any number of naming contexts, provided that they are subtrees of the naming context that names the data store. A data store can contain a mixture of master and replica naming contexts, some or all of which can be replicated to other servers.

Replication

Any naming context held in the data store, including replica naming contexts, can be replicated to another server. The `slapd` daemon can be configured to provide replicated service for a data store using `slurpd`, the directory server update replication daemon. See Chapter 2, "Commands Reference" in the section entitled "`slurpd`" for command-line descriptions about replication.

The load of processing requests generated by directory service clients for the same can be shared between several directory servers. This is done by defining a *replica*, or *slave*, server to provide an alternative access point to the directory service for clients. A master naming context can have more than one replica naming context. FIGURE 4-2 shows a master server with two replica servers.

Replication is the process by which changes in the master data store are propagated to all the replica naming contexts. You can replicate an entire naming context, a subtree, or a particular entry. You can replicate the entire content of an entry or you can specify a subset of attributes to be replicated.

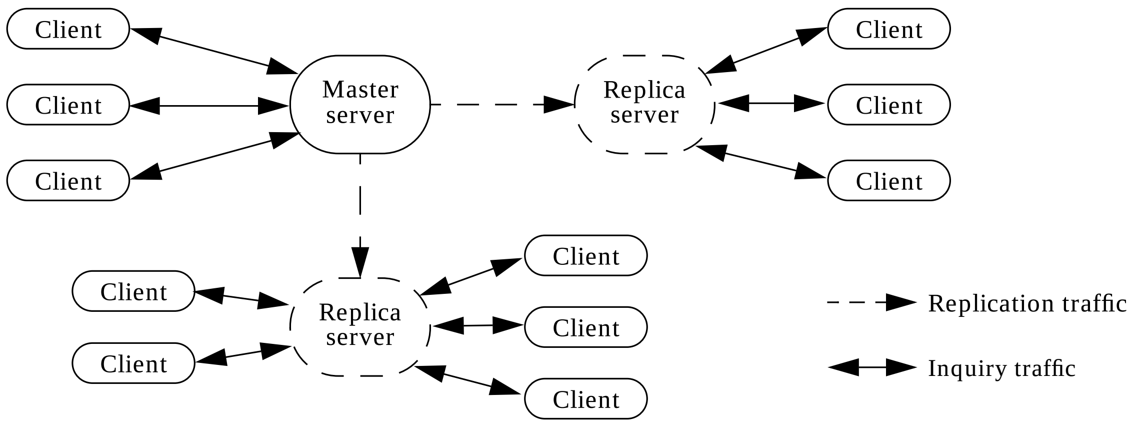

FIGURE 4-2 Master and Replica Servers

Using replication has the following advantages:

- The load on the master server can be reduced by diverting traffic to other servers.
- Copies of data can be stored where it is mostly frequently used to reduce network traffic.
- Multiple copies of the data can be stored, but the data is maintained from a central location.
- Replication can be timed to happen when the network is least busy.

- Only the data that is required by clients of the replica server needs to be copied, if you know the requirements of those clients specifically enough. You may be able to tailor a replica exactly to the needs of a specific client. By reducing the number of entries replicated, network traffic caused by replication updates can be reduced.
- A "public" replica server containing information that is not confidential could be maintained, allowing greater access to this information than you usually allow for other servers. For example, you could create a server containing the email addresses for the sales and support staff who deal with current products but not the research staff working on future products, and make it available to the sales staff of a partner company.

Note – You could provide the same partial view of directory information with appropriate access controls. However, using a partial replica on a dedicated machine ensures that you are not providing access to your entire network. For extra security, you could connect the replica server to your network only while the replication update is in progress.

The negative impacts of using replication are:

- Additional network traffic caused by replication of data. However, though there may be an overall increase in traffic, more of the traffic will be local, so you can avoid known network bottlenecks for inquiry traffic. Also, you can time replication updates for when the network is least busy.
- Information retrieved from replicas may be out of date if replication has not happened since an update, so certain applications may always need to query the master data store.
- You cannot modify a replica. All updates must be performed on the master copy of an entry.

How Replication Works

Information from a master naming context is propagated to a replica by the `slurpd` daemon. The `slurpd` daemon can run permanently, so that updates to directory information are propagated immediately, or you can define a synchronization schedule. You can override the schedule at any time and trigger an immediate synchronization. This is useful if you change a large number of entries and do not want to wait for the next scheduled synchronization.

The `slurpd` daemon uses the LDAP protocol to update a replica naming context. A master naming context for which a replica is defined maintains a replication log. Each time the master naming context is updated, the transaction is recorded in the replication log. When the `slurpd` daemon next runs, it reads the replication log and sends the change to the `slapd` server that holds the replica naming context.

The `slapd` server handles update requests from `slurpd` in the same way that it handles all requests, using the information supplied in the bind request to set the access level granted to `slurpd` requests. To guarantee that all replication updates are completed, `slurpd` must bind with the DN defined when the replica naming context was configured. If a different DN is used, write access for all entries may not be granted.

A replica data store always has a referral pointing to the master data store. If a replica server receives a request to modify an entry, it returns a referral to the client, indicating the master server to be contacted. In some cases, the client software handles the referral automatically and the user need not resubmit the query. Once the modification has been made in the master naming context, the change is sent to the replica naming context the next time the `slurpd` daemon runs.

Example: Replication in the XYZ Corporation

TABLE 4-3 lists the naming contexts in the XYZ Corporation DIT and the servers that store them. In addition, the network management team decide to establish several replica naming contexts:

- All servers will contain a `replica of ou=Boston, o=XYZ, c=US`, for fast access to entries concerning the headquarters of the corporation. Only ussales, eursales, and rowsales get their replicas directly from the boston server. The other servers get a replica of the replica from the server that is closest in the network.
- A second server, eursale2, will hold a complete replica of `ou=Euro-Sales, o=XYZ, c=US`, to share the load on the existing eursales server.
- Each of the servers at the distribution centers will hold complete or partial replicas of the other distribution center naming contexts. For example, the atlanta server will hold a complete replica of `ou=London-Dist, o=XYZ, c=US`, and a partial replica of `ou=Tokyo, o=XYZ, c=US`, containing the information about the distribution center but not about the sales office.

TABLE 4-1 shows the replication strategy for each server in the XYZ Corporation directory service.

TABLE 4-1 Replication Strategy for the XYZ Corporation

Server	Naming Contexts	Replication Status
boston	`ou=Boston, o=XYZ, c=US`	master, replicated to ussales, eursales, and rowsales
ussales	`ou=US-Sales, o=XYZ, c=US`	master
	`ou=Boston, o=XYZ, c=US`	replica from boston, replicated to atlanta and sanfran
eursales	`ou=Euro-Sales, o=XYZ, c=US`	master, replicated to eursale2

TABLE 4-1 Replication Strategy for the XYZ Corporation

Server	Naming Contexts	Replication Status
	`ou=Boston, o=XYZ, c=US`	replica from boston, replicated to eursale2, london, and paris
eursale2	`ou=Euro-Sales, o=XYZ, c=US`	replica from eursales
	`ou=Boston, o=XYZ, c=US`	replica from eursales
rowsales	`ou=RoW-Sales, o=XYZ, c=US`	master
	`ou=Boston, o=XYZ, c=US`	replica from boston
atlanta	`ou=Atlanta-Dist, o=XYZ, c=US`	master, replicated to london and tokyo
	`ou=Boston, o=XYZ, c=US`	replica from ussales
	`ou=London-Dist, o=XYZ, c=US`	replica from london
	`ou=dist, ou=Tokyo, o=XYZ, c=US`	partial replica from tokyo
sanfran	`ou=San-Francisco, o=XYZ, c=US`	master
	`ou=Boston, o=XYZ, c=US`	replica from ussales
london	`ou=London-Dist, o=XYZ, c=US`	master
	`ou=Boston, o=XYZ, c=US`	replica from eursales
	`ou=Atlanta-Dist, o=XYZ, c=US`	replica from atlanta
	`ou=dist, ou=Tokyo, o=XYZ, c=US`	partial replica from tokyo
lonres	`ou=London-RD, o=XYZ, c=US`	master
	`ou=Boston, o=XYZ, c=US`	replica from london
paris	`ou=Paris-Man, o=XYZ, c=US`	master
	`ou=Boston, o=XYZ, c=US`	replica from eursales
tokyo	`ou=Tokyo, o=XYZ, c=US`	master, partially replicated to atlanta and london
	`ou=Boston, o=XYZ, c=US`	replica from rowsales
	`ou=Atlanta-Dist, o=XYZ, c=US`	replica from altanta
	`ou=London-Dist, o=XYZ, c=US`	replica from london

Modifying a Data Store

Use the `ldapmodify` utility to modify an existing data store. You can modify any part of the data store configuration, apart from the distinguished name of the data store naming context.

Initializing a Replica Naming Context

After you have configured a replica naming context, the master and replica data stores must be in the same state, so that the replica can receive replication updates from the master. Use `slapdrepl(1M)` to create an initial replication file and populate the replica using `slurpd`.

Schema

The directory `schema` defines the data that can be stored in the directory. The schema definition is stored in two files:

- `slapd.oc.conf` defines the object classes. These specify the types of entries permitted and their mandatory and optional attributes.
- `slapd.at.conf` contains attribute definition information:
  - Syntax used for any attribute that does not use case-ignore string (cis) syntax
  - Alternate names for some attributes
  - List of naming attributes

  `slapd.at.conf` does not contain the names of all the attributes. The attributes that are not listed in `slapd.at.conf` do not have alternate names and all use case-ignore string syntax.

Note – This section provides definitions for many of the object classes and attributes in the directory schema, including all the most commonly-used items. If you view the schema using the Administration Console, you might see items that are not documented here.

Knowledge Information

A directory server uses knowledge information to pass requests for information to other servers. The knowledge information held by a directory server is a reference to a directory server holding other naming contexts. When a server receives a request for information, it checks whether it can respond to the request using the information in the local data store. If it cannot, it checks the referral defined for the data store, and returns the details of an alternate directory server to the directory client.

The client can then send the request to the other directory server. Some clients contact the alternate server automatically, so the referral mechanism is transparent to users. Other clients return the referral information to the user. See “Example: The XYZ Corporation” for an example of how referrals are used.

Viewing the Schema

You can view the schema using the `imldifsync` command (described in detail in Chapter 2, “Commands Reference,” in the section “Format of Input Files”), though certain objects and attributes cannot be changed.

Modifying the Schema

You can modify the schema by creating new object classes or attributes, or by deleting object classes and attributes. Deleting object classes or attributes is not advisable; some directory entries might use the existing definitions.

Caution – There is no automatic check to ensure that schema modifications do not invalidate entries. Therefore, to minimize the risk of entries becoming invalid, restrict your changes to addition or modification of object classes or attributes.

To create, add, or modify new attributes, use the commands `ldapadd` or `ldapmodify`, as described in Chapter 2, “Commands Reference,” in the sections “`ldapadd`” and “`ldapmodify`.”

Object Classes

This section contains a list of the object classes in the standard schema, explains their purposes, and lists their mandatory and optional attributes. The object classes are described in alphabetical order. The keyword *frozen* after the object class name indicates that this object class is used by a component of Sun Internet Mail Server and that you cannot change the object class definition using the Admin Console. If you change the definition of such an object class, ensure that your changes do not prevent the Sun Internet Mail Server component from using objects of this class.

`account`

Used to define entries representing a user account.

Mandatory attributes: `uid`, `objectClass`

Optional attributes: `description`, `host`, `l`, `o`, `ou`, `seeAlso`

`alias (frozen)`

An alternative name for an object. Objects of class alias must contain the attributes `objectClass` and `aliasedObjectName`.

Mandatory attributes: `aliasedObjectName`, `objectClass`

`ansiOrgObject`

Used to define an entry representing an organization using a code assigned by the American National Standards Institute.

Mandatory attributes: `ansiOrgNumericCode`, `objectClass`

`applicationEntity`

Used to define an entry representing an application entity.

Mandatory attributes: `cn`, `presentationAddress`, `objectClass`

Optional attributes: `description`, `l`, `o`, `ou`, `seeAlso`, `supportedApplicationContext`

`applicationProcess`

Used to define an entry representing an application process.

Mandatory attributes: `cn`, `objectClass`

Optional attributes: `description`, `l`, `ou`, `seeAlso`

`cacheObject`

Used for dynamic objects that reside in cache and that have a limited time-to-live (`ttl`).

Mandatory attributes: `objectClass`

Optional attributes: `ttl`

`certificationAuthority`

Used to define entries representing objects that act as certification authorities.

Mandatory attributes: `authorityRevocationList`, `cACertificate`, `certificateRevocationList`, `crossCertificatePair`, `objectClass`

`country`

Identifies country entries in the directory.

Mandatory attributes: `country`, `objectClass`

Optional attributes: `description`, `l`, `searchGuide`

`device`

Used to define an entry representing a device (for example a modem or CD-ROM drive).

Mandatory attributes: `cn`, `objectClass`

Optional attributes: `description`, `l`, `o`, `ou`, `owner`, `seeAlso`, `serialNumber`

`dNSDomain`

Used to define entries representing a DNS domain.

Mandatory attributes: `dc`, `objectClass`

Optional attributes: `associatedName`, `businessCategory`, `dNSRecord`, `description`, `destinationIndicator`, `facsimileTelephoneNumber`, `internationaliSDNNumber`, `l`, `o`, `physicalDeliveryOfficeName`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `registeredAddress`, `searchGuide`, `seeAlso`, `st`, `streetAddress`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `userPassword`, `x121Address`

`document`

Used to define an entry representing a document.

Mandatory attributes: `documentIdentifier`, `objectClass`

Optional attributes: abstract, audio, authorCN, authorSN, cn, dITRedirect, description, documentAuthor, documentLocation, documentPublisher, documentStore, documentTitle, documentVersion, info, jpegPhoto, keywords, l, lastModifiedBy, lastModifiedTime, manager, o, obsoletedByDocument, obsoletesDocument, ou, photo, seeAlso, subject, uniqueIdentifier, updatedByDocument, updatesDocument

documentSeries

Used to define an entry representing a series of related documents.

Mandatory attributes: cn, objectClass

Optional attributes: description, l, o, ou, seeAlso, telephoneNumber

documentDescription

Used to define an entry that describes a document.

Mandatory attributes: cn, objectClass

Optional attributes: labeledURI, multiLineDescription, owner

domain

Used to define an entry representing a domain.

Mandatory attributes: dc, objectClass

Optional attributes: associatedName, businessCategory, description, destinationIndicator, facsimileTelephoneNumber, internationaliSDNNumber, l, o, physicalDeliveryOfficeName, postOfficeBox, postalAddress, postalCode, preferredDeliveryMethod, registeredAddress, searchGuide, seeAlso, st, streetAddress, telephoneNumber, teletexTerminalIdentifier, telexNumber, userPassword, x121Address

domainRelatedObject

Used to define an entry related to a domain.

Mandatory attributes: associatedDomain, objectClass

dSA

Used to define an entry representing a directory system agent (DSA) or any directory server.

Mandatory attributes: cn, presentationAddress, objectClass

Optional attributes: knowledgeInformation

emailGroup

Used to define an entry representing an electronic mail distribution list that uses aliases(4) format.

Mandatory attributes: cn, objectClass

Optional attributes: authorizedDomain, authorizedSubmitter, dataSource, expandable, mailDeliveryFile, mailDeliveryOption, mailProgramDeliveryInfo, mailHost, ownerDeliveryFile, ownerDeliveryOption, ownerProgramDeliveryInfo, requestsToDeliveryFile, requestsToDeliveryOption, requestsToProgramDeliveryInfo, rfc822AuthorizedSubmitter, rfc822MailMember, rfc822Owner, rfc822UnauthorizedSubmitter, unauthorizedDomain, unauthorizedSubmitter

emailPerson (frozen)

Used to define an entry for a person who uses electronic mail.

Mandatory attributes: cn, objectClass

Optional attributes: assistant, channelName, channelType, dataSource, generationQualifier, freeFormName, homeDirectory, homeFacsimileTelephoneNumber, mail, mailAutoReplyExpirationDate, mailAutoReplyMode, mailAutoReplySubject, mailAutoReplyText, mailAutoReplyTextInternal, mailDeliveryFile, mailDeliveryOption, mailForwardingAddress, mailHost, mailMessageStore, mailProgramDeliveryInfo, mailQuota, objectStatus, preferredRfc822Recipient, reportsTo, rfc822Mailbox, userDefinedAttribute1, userDefinedAttribute2, userDefinedAttribute3, userDefinedAttribute4

fips55Object

Used to define an entry for a location within the US using the FIPS 55 code.

Mandatory attributes: `fips55`, `objectClass`

Optional attributes: `st`

friendlyCountry

Used to allow friendlier naming of country entries than with the object class country. The naming attribute of `object class country`, `countryName`, has to be a 2 letter string as defined in ISO 3166.

Mandatory attributes: `c`, `co`, `objectClass`

Optional attributes: `description`, `searchGuide`

gatewayCCMailUser (frozen)

Used to define an entry representing a user of Lotus cc:Mail.

Mandatory attributes: `objectClass`

Optional attributes: `cCMailAddresses`, `preferredCCMailOriginator`, `preferredCCMailRecipient`

gatewayChannel (frozen)

Used to define an entry representing a Legacy Mail gateway channel.

Mandatory attributes: `channelName`, `objectClass`

Optional attributes: `ackedSequenceNumber`, `channelType`, `currentSequenceNumber`, `maxLastModifiedTime`, `objectStatus`, `seeAlso`, `userPassword`

gatewayDocConvPreference (frozen)

Used to store preferences for document conversion for a gateway user.

Mandatory attributes: `objectClass`

Optional attributes: `docConvPreference`

gatewayLotusNotesUser (frozen)

Used to define an entry representing a user of Lotus Notes.

Mandatory attributes: `objectClass`

Optional attributes: `lotusNotesAddresses`, `preferredLotusNotesOriginator`, `preferredLotusNotesRecipient`

gatewayMail11User (frozen)

Used to define an entry representing a user of Mail-11 (DEC).

Mandatory attributes: `objectClass`

Optional attributes: `mail11Addresses`, `preferredMail11Originator`, `preferredMail11Recipient`

gatewayMrUser (frozen)

Used to define an entry representing a user of the legacy Mail Relay (MR) mail system.

Mandatory attributes: `objectClass`

Optional attributes: `mrAddresses`, `preferredMrOriginator`, `preferredMrRecipient`

gatewayMSMailUser (frozen)

Used to define an entry representing a user of Microsoft Mail.

Mandatory attributes: `objectClass`

Optional attributes: `mSMailAddresses`, `preferredMSMailOriginator`, `preferredMSMailRecipient`

gatewayNGMUser (frozen)

Used to define an entry representing a user of the legacy Novell Groupewise Mail (NGM) mail system.

Mandatory attributes: `objectClass`

Optional attributes: `nGMAddresses`, `preferredNGMOriginator`, `preferredNGMRecipient`

`gatewayNGM70User (frozen)`

Used to define an entry representing a user of the legacy Novell Groupewise Mail 7.0 (NGM70) mail system.

Mandatory attributes: `objectClass`

Optional attributes: `nGM70Addresses`, `preferredNGM70Originator`, `preferredNGM70Recipient`

`gatewayPROFSUser (frozen)`

Used to define an entry representing a user of IBM PROFS.

Mandatory attributes: `objectClass`

Optional attributes: `pROFSAddresses`, `preferredPROFSOriginator`, `preferredPROFSRecipient`

`groupOfNames`

Used to define entries representing an unordered set of names of objects or other groups.

Mandatory attributes: `cn`, `member`, `objectClass`

Optional attributes: `businessCategory`, `description`, `o`, `ou`, `owner`, `seeAlso`

`groupOfUniqueNames`

Used to define entries representing an unordered set of names of objects or other groups. Each name in the set is unique in the directory.

Mandatory attributes: `cn`, `uniqueMember`, `objectClass`

Optional attributes: `businessCategory`, `description`, `o`, `ou`, `owner`, `seeAlso`

`image`

Used to define an entry representing an image.

Mandatory attributes: `cn`, `objectClass`

Optional attributes: `citation`, `copyright`, `imageFiles`, `jpegPhoto`, `keywords`, `multiLineDescription`, `owner`, `predominantColor`

imageFile

Used to define an entry representing a file that contains an image.

Mandatory attributes: cn, objectClass

Optional attributes: colorDepth, documentLocation, fileFormat, fileSize, height, resolution, seeAlso, width

inetOrgPerson

Used to define an entry for a person who uses the Internet and belongs to an organization.

Mandatory attributes: cn, sn, objectClass

Optional attributes: audio, businessCategory, carLicense, departmentNumber, description, destinationIndicator, employeeNumber, employeeType, facsimileTelephoneNumber, givenName, homePhone, homePostalAddress, initials, internationaliSDNNumber, jpegPhoto, l, labeledURI, mail, manager, mobile, ou, pager, photo, physicalDeliveryOfficeName, postOfficeBox, postalAddress, postalCode, preferredDeliveryMethod, registeredAddress, roomNumber, secretary, seeAlso, st, streetAddress, telephoneNumber, teletexTerminalIdentifier, telexNumber, title, uid, x500uniqueIdentifier, userPassword, userCertificate, userCertificate;binary, x121Address

kerberosSecurityObject

Used to define an entry that stores the Kerberos name of an object.

Mandatory attributes: krbName, objectClass

labeledURIObject

Used to define an entry that describes a resource on the network that is identified by a URI.

Mandatory attributes: objectClass

Optional attributes: labeledURI

`locality`

Used to define entries that describe locality.

Mandatory attributes: `objectClass`

Optional attributes: `description`, `locality`, `searchGuide`, `seeAlso`, `st`, `streetAddress`

`nadfADDMD`

Used to define an entry representing an administrative directory domain.

Mandatory attributes: `ad`, `objectClass`

Optional attributes: `businessCategory`, `description`, `destinationIndicator`, `facsimileTelephoneNumber`, `internationaliSDNNumber`, `l`, `lastModifiedTime`, `nadfSearchGuide`, `o`, `physicalDeliveryOfficeName`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `registeredAddress`, `searchGuide`, `seeAlso`, `st`, `streetAddress`, `supplementaryInformation`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `userPassword`, `x121Address`

`nadfApplicationEntity`

Used to define an entry representing an Application Entity.

Mandatory attributes: `cn`, `presentationAddress`, `supportedApplicationContext`, `objectClass`

Optional attributes: `description`, `l`, `o`, `ou`, `seeAlso`, `supportedApplicationContext`

`nationalObject`

Used to define objects associated with a specific country.

Mandatory attributes: `c`, `objectClass`

`organization`

Used to define organization entries in the directory.

Mandatory attributes: `organization`, `objectClass`

Optional attributes: `businessCategory`, `description`, `destinationIndicator`, `facsimileTelephoneNumber`, `internationaliSDNNumber`, `locality`, `physicalDeliveryOfficeName`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `registeredAddress`, `searchGuide`, `seeAlso`, `st`, `streetAddress`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `userPassword`, `x121Address`

organizationalPerson

Used to define entries representing people employed by, or in some way associated with, and organization.

Mandatory attributes: `cn`, `sn`, `objectClass`

Optional attributes: `description`, `destinationIndicator`, `facsimileTelephoneNumber`, `internationaliSDNNumber`, `l`, `ou`, `physicalDeliveryOfficeName`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `registeredAddress`, `seeAlso`, `st`, `streetAddress`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `title`, `userPassword`, `x121Address`

organizationalRole

Used to define entries representing a role or position within an organization. An `organizationalRole` is usually filled by an `organizationalPerson`, but it can also be filled by a nonhuman.

Mandatory attributes: `cn`, `objectClass`

Optional attributes: `description`, `destinationIndicator`, `facsimileTelephoneNumber`, `internationaliSDNNumber`, `l`, `ou`, `physicalDeliveryOfficeName`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `registeredAddress`, `roleOccupant`, `seeAlso`, `st`, `streetAddress`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `x121Address`

organizationalUnit

Used to define entries representing subdivisions of an organization.

Mandatory attributes: `ou`, `objectClass`

Optional attributes: businessCategory, description, destinationIndicator, facsimileTelephoneNumber, internationaliSDNNumber, l, physicalDeliveryOfficeName, postOfficeBox, postalAddress, postalCode, registeredAddress, searchGuide, seeAlso, st, streetAddress, telephoneNumber, teletexTerminalIdentifier, telexNumber, userPassword, x121Address

person

Used to define entries representing people.

Mandatory attributes: cn, sn, objectClass

Optional attributes: description, seeAlso, telephoneNumber, userPassword

residentialPerson

Used to define entries representing a person in the residential environment.

Mandatory attributes: cn, l, sn, objectClass

Optional attributes: businessCategory, description, destinationIndicator, facsimileTelephoneNumber, internationaliSDNNumber, l, physicalDeliveryOfficeName, postOfficeBox, postalAddress, postalCode, preferredDeliveryMethod, registeredAddress, seeAlso, st, streetAddress, telephoneNumber, teletexTerminalIdentifier, telexNumber, userPassword, x121Address

rFC822LocalPart

Used to define entries which represent the local part of RFC822 mail addresses. This treats this part of an RFC822 address as a domain.

Mandatory attributes: dc, objectClass

Optional attributes: associatedName, businessCategory, cn, description, destinationIndicator, facsimileTelephoneNumber, internationaliSDNNumber, l, o, physicalDeliveryOfficeName, postOfficeBox, postalAddress, postalCode, preferredDeliveryMethod, registeredAddress, searchGuide, seeAlso, sn, st, streetAddress, telephoneNumber, teletexTerminalIdentifier, telexNumber, userPassword, x121Address

rfc822MailGroup

Used to define an entry representing a distribution list.

Mandatory attributes: `cn`, `objectClass`

Optional attributes: `associatedDomain`, `autoMgt`, `description`, `destinationIndicator`, `errorsTo`, `facsimileTelephoneNumber`, `internationaliSDNNumber`, `joinable`, `krbName`, `labeledURI`, `mail`, `member`, `memberOfGroup`, `moderator`, `multiLineDescription`, `notice`, `owner`, `physicalDeliveryOfficeName`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `registeredAddress`, `requestsTo`, `rfc822ErrorsTo`, `rfc822RequestsTo`, `seeAlso`, `streetAddress`, `suppressNoEmailError`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `userPassword`, `x121Address`, `xacl`

room

Used to define an entry representing a room.

Mandatory attributes: `cn`, `objectClass`

`Optional attributes: description, roomNumber, seeAlso, telephoneNumber`

service

Used to define an entry representing a service.

Mandatory attributes: `cn`, `objectClass`

Optional attributes: `category`, `dependentUpon`, `destinationIndicator`, `facsimileTelephoneNumber`, `hoursOfOperation`, `internationaliSDNNumber`, `jpegPhoto`, `keywords`, `labeledURI`, `mail`, `multiLineDescription`, `owner`, `physicalDeliveryOfficeName`, `platform`, `postOfficeBox`, `postalAddress`, `postalCode`, `preferredDeliveryMethod`, `product`, `provider`, `ratingDescription`, `ratingTime`, `registeredAddress`, `seeAlso`, `serviceArea`, `serviceRating`, `streetAddress`, `telephoneNumber`, `teletexTerminalIdentifier`, `telexNumber`, `x121Address`

simpleSecurityObject

Used to define an entry containing a user password, for simple authentication.

Mandatory attributes: `userPassword`, `objectClass`

slapdNonLeafObject (frozen)

Used to ensure that all directory entries that have child entries include the objectClass attribute.

Mandatory attributes: objectClass

slapdObject (frozen)

Used to ensure that all directory entries contain the objectClass attribute.

Mandatory attributes: objectClass

strongAuthenticationUser

Used to define an entry for an object participating in Strong Authentication.

Mandatory attributes: userCertificate, objectClass

usCountyOrEquivalent

Used to define an entry representing a US county.

Mandatory attributes: fips55, fipsCountyNumericCode, l, objectClass

Optional attributes: description, lastModifiedTime, nadfSearchGuide, searchGuide, seeAlso, st, streetAddress, supplementaryInformation

usPlace

Used to define an entry representing a location in the US.

Mandatory attributes: fips55, l, objectClass

Optional attributes: description, lastModifiedTime, nadfSearchGuide, searchGuide, seeAlso, st, streetAddress, supplementaryInformation

usStateOrEquivalent

Used to define an entry representing a US state

Mandatory attributes: fipsStateAlphaCode, fipsStateNumericCode, l, st, objectClass

Optional attributes: `description`, `lastModifiedTime`, `nadfSearchGuide`, `searchGuide`, `seeAlso`, `streetAddress`, `supplementaryInformation`

Attributes

All attributes defined in the schema have one of the following syntaxes:

- Distinguished name (`dn`)
- Case-ignore string (`cis`)

  An alphanumeric string, *not* case-sensitive
- Case-exact string (`ces`)

  A case-sensitive alphanumeric string
- Telephone number (`tel`)
- Integer (`int`)
- Binary (`bin`)
- Encrypted (`protected`)

  A value that has been encrypted using `crypt`(1)
- UTC time (`utctime`)

The following list of attributes in the standard schema gives the attribute syntax, any alternative names, and explains how the attribute is used.

`abstract`

Syntax: `cis`

Purpose: A brief description of the document described by the entry.

`ackedSequenceNumber`

Syntax: `ces`

Purpose: A sequence number used during Legacy Mail directory synchronization.

`ad`

Syntax: `cis`

Purpose: The directory administration domain of the object described by the entry.

`aliasedObjectName`

Alternate name: `aliasedEntryName`

Syntax: `dn`

Purpose: The Distinguished Name (DN) of the entry for which the alias entry is an alias.

`ansiOrgNumericCode`

Syntax: `cis`

Purpose: A numeric code identifying the organization described in the entry.

`assistant`

Syntax: `cis`

Purpose: An assistant to the person described by the entry.

`associatedDomain`

Syntax: `cis`

Purpose: The domain with which the object described by this entry is associated.

`associatedName`

Syntax: dn

Purpose: The DN of an entry associated with this entry.

`audio`

Syntax: `bin`

Purpose: Sound information associated with the object described by the entry.

`authorCN`

Syntax: `cis`

Purpose: The common name of the author of a document described by the entry.

`authorSN`

Syntax: `cis`

Purpose: The surname of the author of a document described by the entry.

`authorityRevocationList`

Syntax: `cis`

Purpose: A list of certificates that have been revoked by the certification authority described in the entry, or that the certification authority knows have been revoked by other certification authorities.

`authorizedDomain`

Syntax: `cis`

Purpose: Domain name from which users are authorized to post to the list described by the entry.

`authorizedSubmitter`

Syntax: `cis`

Purpose: A registered user authorized to post messages to the list described by the entry.

`autoMgt`

Syntax: `cis`

Purpose: Whether or not the list described by the entry is managed automatically. The value is `TRUE` or `FALSE`.

buildingName

Syntax: `cis`

Purpose: The name of the building where the object described by the entry resides.

businessCategory

Syntax: `cis`

Purpose: The type of business of the object described by the entry.

cACertificate

Syntax: `bin`

Purpose: The public key of the certification authority described by the entry.

category

Syntax: `cis`

Purpose: The category of the service described by the entry.

cCMailAddress

Syntax: `cis`

Purpose: Used to route email messages through a Lotus cc:Mail channel. It stores a copy of the email addresses in the `preferredCCMailOriginator` and `preferredCCMailRecipient` attributes.

certificateRevocationList

Syntax: `bin`

Purpose: A list of certificates that have been revoked by the certification authority described by the entry, or that the certification authority knows have been revoked by other certification authorities.

`channelName`

Alternate name: `ch`

Syntax: `cis`

Purpose: The name of the Legacy Mail channel for the user described by the entry. Channel names are chosen for users by the system administrator.

`channelType`

Syntax: `ces`

Purpose: The type of the Legacy Mail channel for the user described in the entry. The value must be one of the following:

- 0 for cc:Mail users
- 1 for Microsoft Mail users
- 4 for an Internet mail users, mandatory for routing of messages
- 8 for IBM PROFS users

`citation`

Syntax: `cis`

Purpose: Information related to the source of the image described in the entry.

`colorDepth`

Syntax: `cis`

Purpose: The number of bits required to represent each pixel in the image described by the entry.

`commonName`

Alternate name: `cn`

Syntax: `cis`

Purpose: The full name of the user described by the entry.

`copyright`

Syntax: `cis`

Purpose: The copyright statement for the object described by the entry.

`countryName`

Alternate name: `c`

Syntax: `cis`

Purpose: The name of the country where the object described by the entry resides, or where a parent of the entry resides. Multinational corporations usually use the country of their headquarters as the country of the whole organization.

`crossCertificatePair`

Syntax: `bin`

Purpose: A pair of certificates, containing the public keys of the object described by the entry.

`currentSequenceNumber`

Syntax: `ces`

Purpose: A sequence number used during Legacy Mail directory synchronization.

`dataSource`

Syntax: `cis`

Purpose: The original data source or migration tool for data in the entry.

`domainComponent`

Alternate name: `dc`

Syntax: `cis`

Purpose: The name of the domain described by the entry.

`departmentNumber`

Syntax: `cis`

Purpose: A string identifying the department to which a user described by the entry belongs. The format is a local decision.

`dependentUpon`

Syntax: `cis`

Purpose: Identifies a service, device, or other item that is a prerequisite for the service described in the entry.

`description`

Syntax: `cis`

Purpose: The description of the entry object.

`destinationIndicator`

Syntax: `cis`

Purpose: The country and city addressing information for the object described by the entry.

`dITRedirect`

Syntax: `cis`

Purpose: Indicates that the object described by one entry now has a newer entry in the DIT. The entry containing the redirection attribute should be expired after a suitable grace period.

`dNSRecord`

Syntax: `cis`

Purpose: Used to store DNS record fields.

`docConvPreference`

Syntax: `cis`

Purpose: The preferred method for converting a document sent through the gateway described by the entry.

`documentAuthor`

Syntax: `dn`

Purpose: The author of the document described by the entry.

`documentIdentifier`

Syntax: `cis`

Purpose: A string identifying the document described by the entry.

`documentLocation`

Syntax: `cis`

Purpose: The location of the document described by the entry.

`documentPublisher`

Syntax: `cis`

Purpose: The publisher of the document described by the entry.

`documentStore`

Syntax: `cis`

Purpose: The location where the document described by the entry is stored.

`documentTitle`

Syntax: `cis`

Purpose: The title of the document described by the entry.

`documentVersion`

Syntax: `cis`

Purpose: The version number of the document described by the entry.

`employeeNumber`

Syntax: `cis`

Purpose: A number identifying the person described by the entry.

`employeeType`

Syntax: `cis`

Purpose: Information identifying the type of the employee (for example, Contractor) described by the entry.

`errorsTo`

Syntax: `cis`

Purpose: Distinguished name of an entry to which errors concerning the group described by this entry are sent.

`expandable`

Syntax: cis

Purpose: Whether the membership of the list described by the entry is visible (TRUE or FALSE).

`facsimileTelephoneNumber`

Alternate name: `fax`

Syntax: `tel`

Purpose: The fax telephone number of the object described by the entry.

fileFormat

Syntax: `cis`

Purpose: The file format of the image file described by the entry.

fileSize

Syntax: `cis`

Purpose: The size of the image file described by the entry.

fips55

Syntax: `cis`

Purpose: The FIPS 55 code identifying the place described by the entry.

fipsCountyNumericCode

Syntax: `cis`

Purpose: A numeric code identifying the US county where the object described by the entry resides.

fipsStateAlphaCode

Syntax: `cis`

Purpose: An alphabetic code identifying the US state where the object described by the entry resides.

fipsStateNumericCode

Syntax: `cis`

Purpose: A numeric code identifying the US state where the object described by the entry resides.

`freeFormName`

Syntax: `cis`

Purpose: The name of the person described by the entry.

`generationQualifier`

Syntax: `cis`

Purpose: Generation information, for example, `Senior` or `III`, to qualify the name of the user described by the entry.

`givenName`

Syntax: `cis`

Purpose: The given name of the person described by the entry.

`homeDirectory`

Syntax: `ces`

Purpose: The filesystem location of the home directory of the user described by the entry.

`homeFacsimileTelephoneNumber`

Syntax: `tel`

Purpose: The home fax number of the user described by the entry.

`homePhone`

Syntax: `tel`

Purpose: The home phone number of the user described by the entry.

`homePostalAddress`

Syntax: `cis`

Purpose: The home postal address of the user described by the entry.

`host`

Syntax: `cis`

Purpose: The host used by the object described by the entry.

`hoursOfOperation`

Syntax: `cis`

Purpose: The hours during which the service described by the entry is available.

`imageFiles`

Syntax: `cis`

Purpose: The name of a file containing the image described by the entry.

`info`

Syntax: `cis`

Purpose: General information relating to the object described by the entry.

`initials`

Syntax: `cis`

Purpose: The initials of the person described by the entry.

`internationalISDNNumber`

Syntax: `cis`

Purpose: The ISDN telephone number of the object described by the entry, including country and area codes.

`joinable`

Syntax: `cis`

Purpose: Whether or not users may add themselves to the list described by the entry (`TRUE` or `FALSE`).

`jpegPhoto`

Syntax: `bin`

Purpose: A photograph, in JPEG format, of, or associated with, the object described by the entry.

`keywords`

Syntax: `cis`

Purpose: Keywords associated with the object described by the entry.

`knowledgeInformation`

Syntax: `cis`

Purpose: The knowledge information (references to other directory servers) stored by the DSA described by the entry.

`krbName`

Syntax: `cis`

Purpose: The Kerberos name for the object described by the entry.

`localityName`

Alternate name: `l`

Syntax: `cis`

Purpose: The geographical locality of the object described by the entry.

labeledURI

Alternate name: `labeledURL`

Syntax: `ces`

Purpose: The Uniform Resource Identifier (URI) and label associated with the object described by the entry.

lastModifiedBy

Syntax: `dn`

Purpose: The distinguished name of the user who last modified the object described by the entry. Note that this is not the user who last modified the entry itself.

lastModifiedTime

Syntax: `cis`

Purpose: The date and time when the object described by the entry was last modified. Note that this is not the date and time at which the entry itself was modified.

lotusNotesAddresses

Syntax: `cis`

Purpose: The Lotus Notes electronic mail address of the user described by the entry.

preferredRfc822Originator

Alternate name: `mail`

Syntax: `cis`

Purpose: The advertised electronic mail address, in RFC822 format, of the user described by the entry.

`mail11Addresses`

Syntax: `cis`

Purpose: The Mail-11 electronic mail address of the user described by the entry.

`mailAutoReplyExpirationDate`

Syntax: `cis`

Purpose: At midnight on this date, disable auto-reply to email sent to the user described by the entry. The date must be in UTC format.

`mailAutoReplyMode`

Syntax: `cis`

Purpose: Mode of operation for the auto-reply facility (currently only vacation is supported) for the user described by the entry.

`mailAutoReplySubject`

Syntax: `cis`

Purpose: The subject line of an auto-reply message from the user described by the entry. If it contains the token `$SUBJECT`, it is replaced by the subject line of the incoming message.

`mailAutoReplyText`

Syntax: `cis`

Purpose: The body of an auto-reply message from the user described by the entry. If the text contains the tokens `$SUBJECT` or `$BODY`, they are replaced by the subject line or body from the incoming message. Use `$` as the line-separator.

`mailAutoReplyTextInternal`

Syntax: `cis`

Purpose: The body of an auto-reply message from the user described by the entry, for use within the organization. If the text contains the tokens `$SUBJECT` or `$BODY`, they are replaced by the subject line or body from the incoming message. Use `$` as the line-separator.

`mailDeliveryFile`

Syntax: `ces`

Purpose: The name of a file. Mail delivered to the user whose entry contains this attribute is appended to this file.

`mailDeliveryOption`

Syntax: `cis`

Purpose: Mandatory attribute to allow routing and delivery of messages; delivery routing options include:

- `mailbox`–Deliver mail to the message store mailbox specified by the `mailStore` attribute
- `shared`–Deliver mail to the message store shared-mailbox specified by the `mailStore` attribute
- `native`–Deliver mail to a mail file directory specified in a UNIX file system mailbox
- `autoreply`–Deliver mail to an auto-reply facility (for example, vacation mail)
- `program`–Deliver mail to the UNIX system program specified by the `mailProgramDeliveryInfo` attribute
- `forward`–Forward mail to the address specified by the `mailForwardingAddress` attribute
- `file`–Append mail to the file specified by the `mailDeliveryFile` attribute

Email received by the user described by the entry is delivered according to the options selected.

`mailFolderMap`

Syntax: `cis`

Purpose: Is the message store for a user's mail folders. The value must be one of the following:

- UNIX V7–UNIX V7 message store (also known as the `/var/mail` message store)
- Sun-MS–Sun Internet Mail message store

`mailForwardingAddress`

Syntax: `cis`

Purpose: Forward mail received by the user described by the entry to the specified email address (RFC-822 format).

`mailHost`

Syntax: `cis`

Purpose: The hostname of the `SMTP/MIME` mail server of the user described by the entry, including the full domain name. This hostname is a mandatory attribute for message routing and delivery, or the mail server hostname responsible for expanding the list.

`mailMessageStore`

Syntax: `ces`

Purpose: The filesystem location for the user's inbox as described by the attribute.

`mailProgramDeliveryInfo`

Syntax: `ces`

Purpose: One or more commands, with arguments, to be executed when a message is delivered to the user whose entry contains this attribute if the attribute `mailDeliveryOptions` contains the value program. This attribute is enabled if the mail delivery option='program.'

mailQuota

Syntax: `cis`

Purpose: The maximum size (in bytes) of the message store of the user described by the entry. A value of-1 denotes an unlimited message store.

manager

Syntax: `dn`

Purpose: The Distinguished Name (DN) of the manager of the person or object described by the entry.

maxLastModifiedTime

Syntax: `cis`

Purpose: A time stamp used during Legacy Mail directory synchronization.

member

Syntax: `dn`

Purpose: The distinguished name of a member of the distribution list described by the entry.

memberOfGroup

Syntax: `cis`

Purpose: The distinguished name of a list that has this object as a member.

mobiletelephoneNumber

Alternate name: `mobile`

Syntax: `tel`

Purpose: The telephone number of the mobile phone used by the person described in the entry.

`moderator`

Syntax: `cis`

Purpose: The DN of a moderator for the list described by the entry.

`mrAddresses`

Syntax: `cis`

Purpose: The address of a user of the MR mail system.

`mSMailAddresses`

Syntax: `cis`

Purpose: Used to route email messages through a Microsoft Mail channel. It stores a copy of the email addresses in the `preferredMSMailOriginator` and `preferredMSMailRecipient` attributes.

`multiLineDescription`

Syntax: `cis`

Purpose: A multi-line description of the object described by the entry.

`nadfSearchGuide`

Syntax: `cis`

Purpose: Information to facilitate searching for information contained in the entry.

`nGM70Addresses`

Syntax: `cis`

Purpose: The electronic mail address of a user of the NGM70 mail system.

nGMAddresses

Syntax: `cis`

Purpose: The electronic mail address of a user of the NGM mail system.

notice

Syntax: cis

Purpose: The notice for the list described by the entry.

objectClass

Syntax: `cis`

Purpose: The object class of the type of entry.

objectStatus

Syntax: `cis`

Purpose: Used during Legacy Mail directory synchronization to denote a deleted entry.

obsoletedByDocument

Syntax: `cis`

Purpose: Information identifying a document that makes the document described in the entry obsolete.

obsoletesDocument

Syntax: `cis`

Purpose: Information identifying a document that is made obsolete by the document described in the entry.

`organizationName`

Alternate name: `o`

Syntax: `cis`

Purpose: The name of the organization to which the object described by the entry belongs.

`organizationalUnitName`

Alternate name: `ou`

Syntax: `cis`

Purpose: The name of the organizational unit to which the object described by the entry belongs.

`owner`

Syntax: `dn`

Purpose: The distinguished name of an entry describing the person responsible for the distribution list described by the entry.

`ownerDeliveryFile`

Syntax: `ces`

Purpose: The name of file to which mail addressed to the owner of the distribution list described by the entry is appended.

`ownerDeliveryOption`

Syntax: `ces`

Purpose: Delivery options for mail addressed to `owner`-*listname*. The values must be one of the following:

- `mailbox`–Deliver mail to a message store mailbox
- `shared`–Deliver mail to a message store shared mailbox
- `native`–Deliver mail to a UNIX filesystem mailbox

- `autoreply`–Deliver mail to the `autoreply` facility, such as a vacation mailer
- `program`–Deliver mail to the UNIX program specified as the value of the attribute `ownerProgramDeliveryInfo`
- `forward`–Forward mail to another recipient
- `file`–Append mail to the file specified as a value of the attribute `ownerDeliveryFile`

`ownerProgramDeliveryInfo`

Syntax: `cis`

Purpose: Mail addressed to the owner of a distribution list is delivered to this program. Specifies one or more commands, with arguments, to use in program delivery. Use `$` as the line-separator.

`pagertelephonenumber`

Alternate name: `pager`

Syntax: `tel`

Purpose: The telephone number of the pager of the person described by the entry.

`personalTitle`

Syntax: `cis`

Purpose: The title of the person described by the entry, for example, Doctor or Ms.

`photo`

Syntax: `bin`

Purpose: A photograph of, or associated with, the object described by the entry.

`physicalDeliveryOfficeName`

Syntax: `cis`

Purpose: The mailstop of the object described by the entry.

platform

Syntax: cis

Purpose: The operating system on which the service described by the entry is provided.

postalAddress

Syntax: cis

Purpose: The postal address of the object described by the entry.

postalCode

Syntax: cis

Purpose: The postal code of the object described by the entry.

postOfficeBox

Syntax: cis

Purpose: The post office box of the object described by the entry.

predominantColor

Syntax: cis

Purpose: The predominant color uses in the image described by the entry.

preferredCCMailOriginator

Syntax: cis

Purpose: The email address for routing through a Lotus cc:Mail channel.

preferredCCMailRecipient

Syntax: cis

Purpose: The native Lotus cc:Mail address.

`preferredDeliveryMethod`

Syntax: `cis`

Purpose: Preferred delivery method for communication with the object described by the entry.

`preferredLotusNotesOriginator`

Syntax: `cis`

Purpose: The email address used for routing through a Lotus Notes channel.

`preferredLotusNotesRecipient`

Syntax: `cis`

Purpose: The native Lotus Notes mail address.

`preferredMail11Originator`

Syntax: `cis`

Purpose: The email address used for routing through a Mail-11 channel.

`preferredMail11Recipient`

Syntax: `cis`

Purpose: The native Mail-11 mail address.

`preferredMrOriginator`

Syntax: `cis`

Purpose: The email address used for routing through a Mail Relay (MR) channel.

`preferredMrRecipient`

Syntax: `cis`

Purpose: The native MR mail address.

`preferredMSMailOriginator`

Syntax: `cis`

Purpose: The email address for routing through a Microsoft Mail channel.

`preferredMSMailRecipient`

Syntax: `cis`

Purpose: The native Microsoft Mail address.

`preferredNGM70Originator`

Syntax: `cis`

Purpose: The email address used for routing through a Novell Groupewise Mail 7.0 (NGM70) channel.

`preferredNGM70Recipient`

Syntax: `cis`

Purpose: The native NGM70 address.

`preferredNGMOriginator`

Syntax: `cis`

Purpose: The email address used for routing through a Novell Groupewise Mail (NGM) channel.

`preferredPROFSOriginator`

Syntax: `cis`

Purpose: The email address for routing through an IBM PROFS channel.

`preferredPROFSRecipient`

Syntax: `cis`

Purpose: The native IBM PROFS address.

`preferredRfc822Recipient`

Syntax: `cis`

Purpose: The user's canonical email address (RFC-822 format).

`presentationAddress`

Syntax: `cis`

Purpose: The presentation address of the object described by the entry.

`product`

Syntax: `cis`

Purpose: A product used to provide the service described by the entry.

`pROFSAddresses`

Syntax: `cis`

Purpose: Used to route email messages through an IBM PROFS channel. It stores a copy of the email addresses in the `preferredPROFSOriginator` and `preferredPROFSRecipient` attributes.

`provider`

Syntax: `cis`

Purpose: The entity that provides the service described in the entry.

`ratingDescription`

Syntax: `cis`

Purpose: Describes the rating given to a service.

`ratingTime`

Syntax: `cis`

Purpose: The date at which a service rating occurred.

`registeredAddress`

Syntax: `cis`

Purpose: The registered postal address of the entity described by the entry.

`reportsTo`

Syntax: `cis`

Purpose: The name of the manager of the user described by the entry.

`requestsTo`

Syntax: `cis`

Purpose: The distinguished name of the entity to which requests to be added to the distribution list described by the entry are sent.

`requestsToDeliveryFile`

Syntax: `ces`

Purpose: The name of a file to which requests to be added to the distribution list described by the entry are appended.

`requestsToDeliveryOption`

Syntax: `cis`

Purpose: One or more delivery options for mail addressed to *listname*-request:

- `mailbox`–Deliver mail to a message store mailbox
- `shared`–Deliver mail to a message store shared-mailbox
- `native`–Deliver mail to a UNIX filesystem mailbox
- `autoreply`–Deliver mail to an auto-reply facility (for example, vacation mail)
- `program`–Deliver mail to a UNIX system program
- `forward`–Forward mail to another address
- `file`–Append mail to a file

`requestsToProgramDeliveryInfo`

Syntax: `ces`

Purpose: Mail addressed to *listname*-request is delivered to this program. Specifies one or more commands, with arguments, to use in program delivery. Use `$` as the line-separator.

`resolution`

Syntax: `cis`

Purpose: The resolution of the image contained in the file described by the entry.

`rfc822AuthorizedSubmitter`

Syntax: `cis`

Purpose: The email addresses of users authorized to post to the list.

`rfc822ErrorsTo`

Syntax: `cis`

Purpose: The email address that is notified when mail sent to a distribution list is undeliverable.

`rfc822Mailbox`

Syntax: `cis`

Purpose: Stores all the email addresses (RFC-822 format) defined for the user. It stores a copy of the email addresses in the mail and `preferredRfc822Recipient` attributes.

`rfc822MailMember`

Syntax: `cis`

Purpose: Stores the email addresses (RFC-822 format) defined for members of the list.

`rfc822Owner`

Syntax: `cis`

Purpose: The email addresses of the owner of the list.

`rfc822RequestsTo`

Syntax: `cis`

Purpose: The email address to which requests to be added to the list are sent.

`rfc822UnauthorizedSubmitter`

Syntax: `cis`

Purpose: The email addresses of users not permitted to post to the list.

`roleOccupant`

Syntax: `cis`

Purpose: Information identifying the object or person fulfilling the role described by the entry.

roomNumber

Syntax: `cis`

Purpose: The number of the room where the object described by the entry is located.

searchGuide

Syntax: `cis`

Purpose: Information to facilitate searching for information contained in the entry.

secretary

Syntax: `dn`

Purpose: The Distinguished Name (DN) of the secretary of the person or organization described by the entry.

seeAlso

Syntax: `dn`

Purpose: The DN of an entry that contains information that is also of interest to anyone interested in the object described by this entry.

serialNumber

Syntax: `cis`

Purpose: The serial number of the device described by the entry.

serviceArea

Syntax: `cis`

Purpose: Defines a type of service.

`serviceRating`

Syntax: `cis`

Purpose: Provides a rating of a service.

`surname`

Alternate name: `sn`

Syntax: `cis`

Purpose: The surname of the person described by the entry.

`stateOrProvinceName`

Alternate name: `st`

Syntax: `cis`

Purpose: The name of the state, province, or geographical area within a country where the object described by the entry resides.

`streetAddress`

Alternate name: `street`

Syntax: `cis`

Purpose: A street name and number.

`subject`

Syntax: `cis`

Purpose: The subject of the document described by the entry.

`supplementaryInformation`

Syntax: `cis`

Purpose: Additional information concerning the object described by the entry.

`supportedApplicationContext`

Syntax: `cis`

Purpose: An application context supported by the application entity described by the entry.

`suppressNoEmailError`

Syntax: `cis`

Purpose: Prevent delivery of `No Email` errors (`TRUE` or `FALSE`).

`telephoneNumber`

Syntax: `tel`

Purpose: Telephone number (in international format).

`teletexTerminalIdentifier`

Syntax: `cis`

Purpose: The teletex terminal identifier and, optionally, parameters for a teletex terminal associated with the object described by the entry.

`telexNumber`

Syntax: `cis`

Purpose: Telex number, country code and answerback code of a telex terminal. Dollar-separated string.

`textEncodedORaddress`

Syntax: `cis`

Purpose: The X.400 electronic mail originator/recipient address (`ORAddress`) of the user described in the entry.

`title`

Syntax: `cis`

Purpose: The title of the person described by the entry, for example, Doctor, or Ms.

`ttl`

`Syntax: cis`

Purpose: Specifies the time-to-live (`ttl`) of a cached object.

`unauthorizedDomain`

Syntax: `cis`

Purpose: A domain name from which users are not permitted to post to the list.

`unauthorizedSubmitter`

Syntax: `cis`

Purpose: The registered users not permitted to post messages to the list.

`uniqueIdentifier`

Syntax: `cis`

Purpose: A unique identifier for the object described by the entry.

`updatedByDocument`

Syntax: `cis`

Purpose: Information identifying a document that updates the document described by the entry.

`updatesDocument`

Syntax: `cis`

Purpose: Information identifying a document that is updated by the document described by the entry.

`userCertificate`

Syntax: `bin`

Purpose: A certificate containing the public key of the user described by the entry.

`userCertificate;binary`

Syntax: `bin`

Purpose: A certificate containing the public key of the user described by the entry, in binary format.

`userDefinedAttribute1`

Syntax: `cis`

Purpose: Attribute for use by the user.

`userDefinedAttribute2`

Syntax: `cis`

Purpose: Attribute for use by the user.

`userDefinedAttribute3`

Syntax: `cis`

Purpose: Attribute for use by the user.

`userDefinedAttribute4`

Syntax: `cis`

Purpose: Attribute for use by the user.

`userPassword`

Syntax: `protected`

Purpose: The password of the user described by the entry that is used to gain access to the entry.

`userid`

Alternate name: `uid`

Syntax: `cis`

Purpose: The user ID of the user described by the entry; used to enable routing of mandatory messages and delivery.

`width`

Syntax: `cis`

Purpose: The width of the image contained in the image file described by the entry.

`x121Address`

Syntax: `cis`

Purpose: An address as defined by ITU Recommendation X.121.

`x500uniqueIdentifier`

Syntax: `cis`

Purpose: A unique identifier for the object described by the entry.

Access Control

Access controls determine who has access (permission) to a given directory entry, and what level of access is granted. The following section, "Configuring Access Control," explains how to design an access control policy for your directory. The following sections explain how to add, modify, and delete access control rules, using the commands described in Chapter 2, "Commands Reference," in the sections "`ldapadd`", "`ldapdelete`", and "`ldapmodify`."

An access control rule defines the level of access (sets the permissions) to specific directory information given to a particular user. The two stages used to define a new access control rule are:

- Specify the directory information to which the rule applies. This is the information that you want to protect.
- Specify the level of access granted to each user for this information.

Access control rules are hierarchical, with the most specific rules listed first, followed by more general rules. At a given level of the hierarchy, the action of the rule is limited by the access controls at the level above. At any level, the term *All entries* means all entries for which a specific access control rule has not already been set.

Configuring Access Control

Access to information in the directory is controlled by a set of rules that determine what permissions a user requires in order to access an entry or an attribute. The levels of permission for directory information are:

- None–You are not permitted to access the entry at all, and cannot see information indicating that the entry exists.
- Compare–You can compare the value of a given attribute with a value you supply, but you cannot read the attribute value. This is used when checking passwords.
- Search–You can read the distinguished name of an entry, and you can search for entries based on the existence of an attribute or attribute value. You cannot necessarily read the attribute value.

- Read–You can read the value of an attribute within an entry.
- Write–You can write information into an entry or attribute; that is, you can modify or delete an attribute value, attribute, or whole entry.

Note – If you have permission to read the attributes of an entry, you also have permission to search and compare.

Access control rules define which users are granted which permissions for a given set of entries or attributes. For example, you can give a privileged user read permission for all attributes except password in all entries, and compare permission for password attributes.

Access control rules for any set of entries can be defined by:

- All entries
- A distinguished name-based regular expression (see "Specifying a Distinguished Name")
- An LDAP filter (see "Specifying an LDAP Filter")
- Presence of a particular attribute

You can define access control rules that apply to the person described by an entry (using the keyword *self*), so that, for example, only you can change your own password. You can also define access control rules that apply to any user (using the keyword *everyone* or *).

The access control rules are applied in sequence, so the order in which they are listed is important. You must state the most specific rules first, with more general rules afterward. "Configuring Access Control" explains how to define an access control rule using the configuration tool, and how to specify the order of rules.

For example, you could define the following access control rules:

- Users have write access to their own password attribute, but only compare access to the passwords of other users.
- A user whose entry contains the attribute value `locality=San Francisco` has read access to all other entries that contain the attribute value `locality=San Francisco`, but cannot read the password attribute value.

The default access controls defined at installation are as follows:

- All users have compare access to the values of the attribute `userPassword`. To change the value of the `userPassword` attribute, you must bind with the DN of the entry containing the attribute, that is, the password can only be changed by the owner of the entry.

- Everyone has read access to the following attributes: `cn`, `dataSource`, `homeDirectory`, `messageStore`, `messageStoreSizeQuota`, `mail`, `mailServer`, `objectStatus`, `preferredRfc822Recipient`, `rfc822Mailbox`, and `uid`.
- Any user whose DN contains the attribute member has write access to the member and entry attributes of any entry containing the attribute joinable with value `TRUE`. These attributes are also writable by any user who binds with the DN of the entry.
- Anyone binding with the DN of an entry has write access to that entry. Everyone else has read access only.
- The system administrator always has complete access to all attributes in all entries. You cannot change the access granted to the administrator, which ensures that there is always at least one user who has access to every entry in the directory.

These rules are applied in order, starting with the most specific followed by the more general rules. The following table shows how the default access controls are defined in the directory server configuration file (for information about configuration files, see Chapter 2, "Commands Reference," in the sections "`slapd`" and "`slurpd`").

TABLE 4-2 Permission Attributes for User Access

```
access to attrs=userPassword
                 by self write
                 by * compare

access to attrs=cn, dataSource, homeDirectory, messageStore,
messageStoreSizeQuota, mail, mailServer, objectStatus,
preferredRfc822Recipient, rfc822Mailbox, uid
                 by self read
                 by * read
access to filter="joinable=TRUE" attrs= member, entry
                 by dnattr=member self write
access to *
                 by self write
                 by * read
```

TABLE 4-2 lists permission attributes for user access.

All directory interactions begin with a bind. The information used to establish the bind is also used to determine the permission level at which you are granted access to the directory. All further interaction with the directory for the duration of the bind is regulated by this permission level.

Directory Structure

Directory information is stored in a *data store*. A server can contain more than one data store. A data store is the physical location where a naming context is held, and identified by the distinguished name of the naming context it stores. A data store can hold more than one naming context and can have access restrictions as those defined previously.

Information in the directory is organized in a tree structure, called the *Directory Information Tree* (DIT). The structure of a DIT usually reflects very closely the structure of the information it contains. For example, a directory containing entries for people in a corporation can be organized by division or by location. In general, DIT structures are organizational, geographical, or include both organizational and geographical factors.

When dividing the DIT into data stores to be held on individual servers, you need to take account of the following:

- A server can hold more than one data store, with certain limitations:
  - You cannot search more than one data store at a time.
  - Alias definitions are local to a data store.
- A server can hold both master and replica naming contexts, provided they are in separate data stores. See "Replication" for information about defining a replication strategy.

Example: The XYZ Corporation

Let's assume a scenario in which the XYZ Corporation is a pharmaceutical company, with headquarters in Boston, USA. They have two manufacturing operations, one in San Francisco and one in Paris, and three distribution centers, in Atlanta, London, and Tokyo. There are two research groups, in London and San Francisco, located with the other XYZ divisions in those cities. The Sales organization has three divisions: Europe, US&C (US and Canada), and the Rest of the World (RoW). FIGURE 4-3 shows the functional structure of XYZ Corporation.

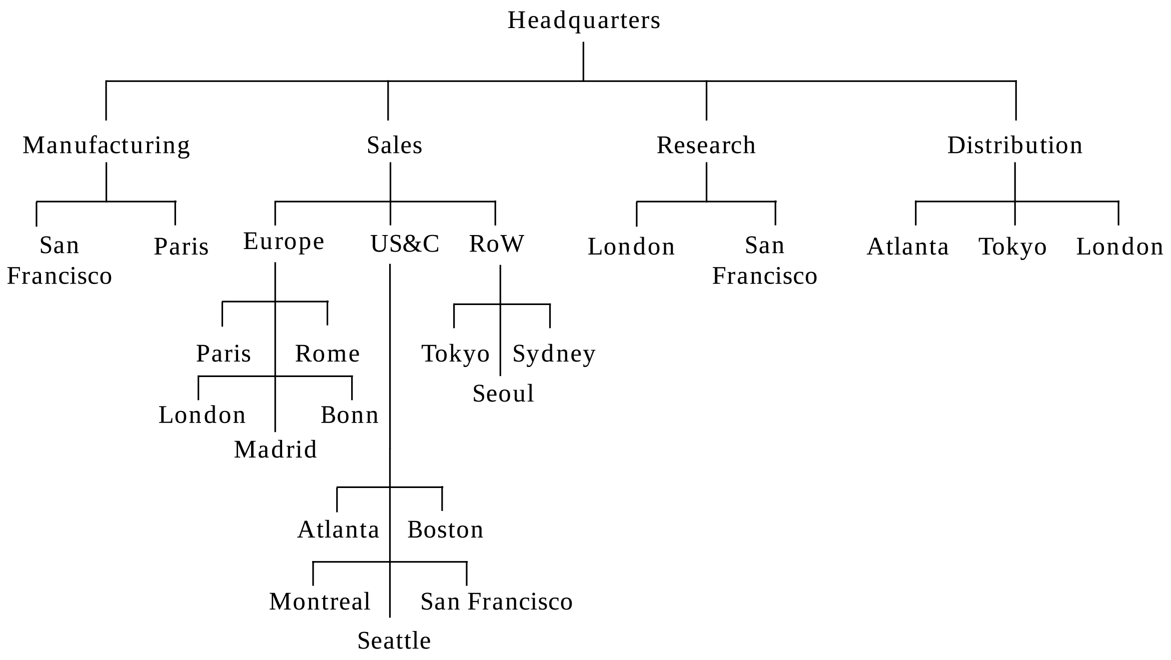

FIGURE 4-3 Functional Structure of XYZ Corporation

FIGURE 4-4 shows the geographical structure of XYZ Corporation.

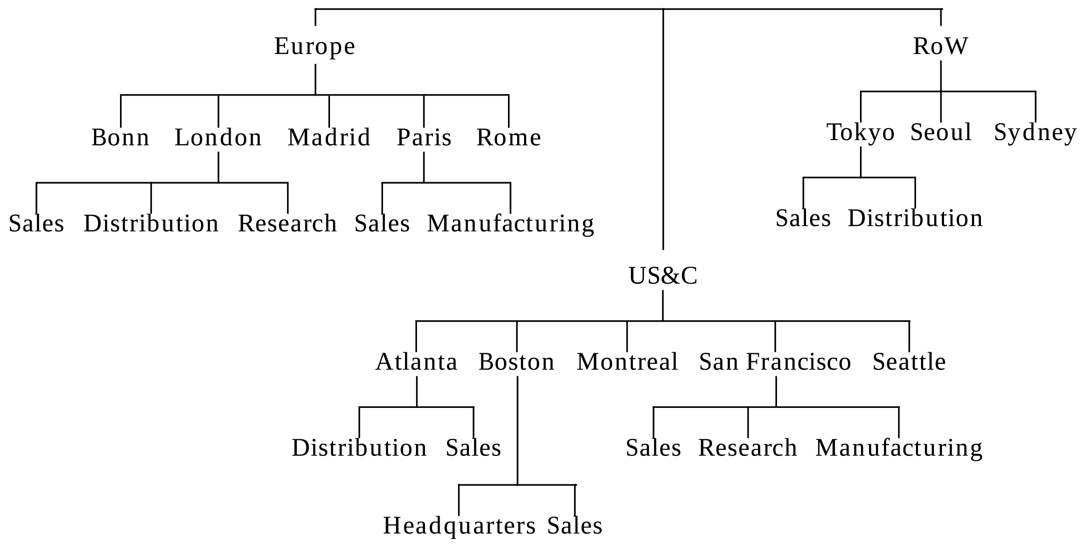

FIGURE 4-4 Geographical Structure of XYZ Corporation

As is common with many organizations, neither an organizational DIT structure nor a functional DIT structure completely meets the directory structure needs of XYZ Corporation, so the network management team decides to combine functional and geographical factors, and to take into account the different usage patterns within the different departments. The result is the DIT structure shown in FIGURE 4-5.

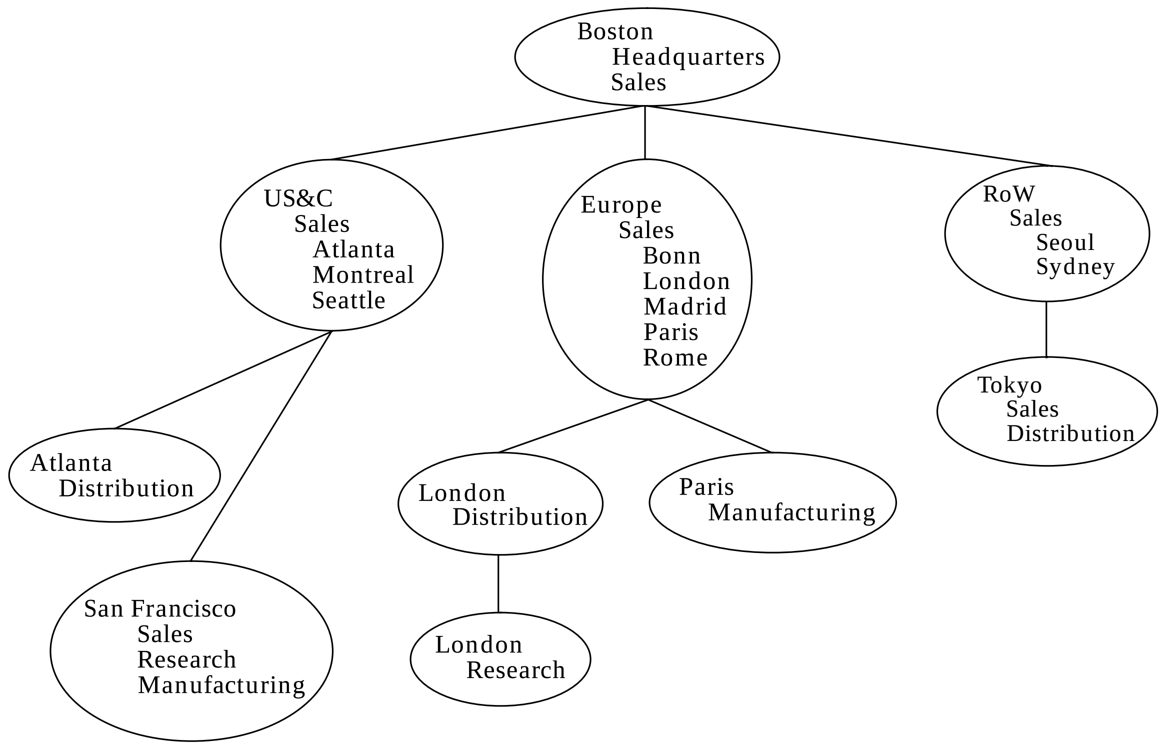

FIGURE 4-5 DIT Structure for XYZ Corporation

In this DIT structure there are ten naming contexts. Each naming context contains entries that are related to a particular geographical or functional area. Each naming context can be stored on a different host, which, given that much of the enquiry traffic is expected to be local to a server, reduces the network traffic. TABLE 4-3 lists the distinguished names of the naming contexts in the DIT structure, and the name of the server holding each data store.

TABLE 4-3 XYZ Corporation Naming Contexts

Naming Context	Server
`ou=Boston, o=XYZ, c=US`	boston
`ou=US-Sales, o=XYZ, c=US`	ussales
`ou=Atlanta-Dist, o=XYZ, c=US`	atlanta
`ou=San-Francisco, o=XYZ, c=US`	sanfran
`ou=Euro-Sales, o=XYZ, c=US`	eursales

TABLE 4-3 XYZ Corporation Naming Contexts

Naming Context	Server
`ou=London-Dist, o=XYZ, c=US`	london
`ou=London-RD, o=XYZ, c=US`	lonres
`ou=Paris-Man, o=XYZ, c=US`	paris
`ou=RoW-Sales, o=XYZ, c=US`	rowsales
`ou=Tokyo, o=XYZ, c=US`	tokyo

A referral system ensures that if an entry cannot be found locally, the directory server can pass the request to another directory server.

TABLE 4-4 shows the referrals defined on each server.

TABLE 4-4 XYZ Corporation Referrals

Server	Referral
boston	none
ussales	boston
atlanta	ussales
sanfran	ussales
eursales	boston
london	eursales
lonres	eursales
paris	eursales
rowsales	boston
tokyo	rowsales

CHAPTER **5**

Message Access and Store Configuration

The `ims.cnf` file is the configuration file for the SIMS Message Store and Message Access components. `ims.cnf` contains configuration parameters for the Message Store and Message Access utilities.

The `ims.cnf` File

In order to make configuration changes to the `ims.cnf` file, you can either edit the file manually or use the SIMS administration console. It is recommended that you use the SIMS administration console rather than editing the `ims.cnf` file manually.

Any changes made to the Message Store paths should be made when no Message Store utilities are running.

Each entry in the `ims.cnf` file has the form:

```
ims-parameter-name: value
```

The parameters are broken down into the following categories: Message Store paths, Message Store file system, Message Store delivery, and Message Access. The parameters are described in the following sections.

Message Store Paths

TABLE 5-1 describes the parameters for the Message Store paths.

TABLE 5-1 Message Store Paths

Parameter	Description
`ims-user-root`	Path to the per-user files. The default value is `/var/opt/SUNWmail/ims/user`.
`ims-index-root`	Path to the index files. The default value is `/var/opt/SUNWmail/ims/index`.
`ims-data-root`	Path to the data files. The default value is `/var/opt/SUNWmail/ims/data`.
`ims-hash-root`	Path to the hashing indices. This path is currently unused but must exist. The default value is `/var/opt/SUNWmail/ims/hash`.
`ims-adm-root`	Path to where the files and reports are written by the `imcheck` utility. Also path where internal lock files and Legato (Solstice Backup) directory reside. The default value is `/var/opt/SUNWmail/ims/adm`.
`ims-shared-root`	Path to shared mailboxes. The default value is `/var/opt/SUNWmail/ims/shared`.

Message Store File System

TABLE 5-2 describes the parameters for the Message Store file system.

TABLE 5-2 Message Store File System

Parameter	Description
`ims-owner`	Solaris owner of all the Message Store files. The default value is `inetmail`.
`ims-init-interval`	Number of days to create at initialization. The default value is 30.
`ims-augment-interval`	Number of days to create at one time. The default value is 30.

Message Store Delivery

TABLE 5-3 describes the parameters for the Message Store delivery utility (`ims_master`).

TABLE 5-3 Message Store Delivery

Parameter	Description
`ims-parse-level`	The level of parsing for incoming messages. 1=POP-only store and 3=IMAP or POP3. The level must not go from 3 to 1. The default value is 3.
`ims-quota`	Specifies whether per-user quotas are enforced or not. The default is `OFF`.
`ims-default-quota`	The default quota in bytes for users. This value is used if the information is not provided in the directory. The default value is 20000000.

Message Access

TABLE 5-4 describes the parameters for the Message Access utility (`imaccessd`).

TABLE 5-4 Message Access

Parameter	Description
`ims-ldap-server`	Name of the LDAP server used for authentication. The default value is `localhost`.
`ims-mail-host`	The default domain for parsing email addresses when no `@domain` is present. The default value is `localhost`.
`ims-basedn`	Base node in the LDAP tree used to search for users at login time.
`ims-varmail`	`ON` specifies that users can access mailboxes in the `/var/mail` format in addition to the SIMS Message Store format. The default value is `OFF`.
`ims-maxconnections`	Number of connections that can be simultaneously supported by the message access server. The default value is `10000`.

TABLE 5-4 Message Access

Parameter	Description
`ims-proxy`	Specifies the proxy behavior of the message access server. `OFF` specifies that the proxy is disabled—only local users have access. `ON` specifies that the proxy is enabled—local and proxy users have access. `ONLY` specifies that the server is only a proxy—no local store access.
`ims-caps-proxy`	Specifies the IMAP4 capabilities advertised by imaccessd when the proxy behavior is `ON` or `ONLY`. The default value is `IMAP4 IMAP4rev1`.
`ims-bind-address`	Specifies the interface (IP address or hostname) and ports that are listened to during POP and IMAP connections. The value of `ims-bind-address` is in the form: `[`*hostname* `| *][(pop3=`*port1*`[,`*port2*`, ...]:imap=`*port3*`[,`*port4*`, ...])]` This parameter can appear multiple times in the `ims.cnf` file. By default, or if the hostname is *, `imaccessd` listens to all the addresses supported by the server. If no service or ports are specified, the default ports are fetched from `/etc/services`. Specifying a port as 0 denotes that the service (IMAP or POP3) is not supported on that particular server. `pop3s` and `imaps` ports (SSL is used) can be defined. The following example provides POP3 service on ports 110 and 109, IMAP service on the default port as specified in `/etc/services`, and no POP3s or IMAPs (secure) services. POP3 and IMAP service will be provided only on the interface identified by `mail1` and not on the other interfaces on the server. `mail1.com.net(pop3=110,109:pop3s=0;imaps=0)`

CHAPTER 6

System Architecture

This chapter describes the following components of the Sun Internet Mail message handling system:

- Message Access Services
- Message Store
- Internet Message Transfer Agent
- Sun Directory Services
- Administration Services
- Sun Messaging Connectivity Services

Message Access Services

Message Access Services refers to the protocol servers, software drivers, and libraries that support client access to the message store. The key to this component is the Internet Message Access Protocol version 4 (IMAP4), implemented via the c-client Mail API library. This component is also responsible for the Post Office Protocol version 3 (POP3). The key features of this component are:

- IMAP4 Revision 1–This protocol has been extended to optimize network usage and improve low-bandwidth performance.
- Advanced POP3–In addition to the mandatory POP3 command set, Message Access Services supports the optional `TOP` and `UIDL` commands. The `UIDL` implementation is based on IMAP4 Universal Identifiers.
- Orthogonal Message Store Access–Multiple access protocols (IMAP and POP3) are supported from a common message store. Similarly, both message stores (Solaris Mailbox Format and Sun Message Store) support both access protocols. It is also possible for the IMAP server to place the user's inbox in one store, and personal mail folders in the other.

- Support for Sun Mailtool V3 Attachment Format–Documents received in Sun V3 format are automatically converted in the message stores to MIME.

Component Architecture

At the heart of the component is the Sun Internet Message Access Services Library. This is a general-purpose message access API, derived from the c-client API Library developed at the University of Washington.

The following submodules are TCP protocol servers. They are independent of each other, and can be freely included or excluded from a particular consolidation.

- Internet Message Access Protocol (IMAP) version 4rev1 Server. Provided for the server side of the standard IMAP4rev1 (RFC 2060) access protocol, this includes full compatibility with IMAP4 (RFC 1730) and IMAP2 (RFC 1176), and the IMAP2bis extensions published by the University of Washington.
- Post Office Protocol (POP) version 3 Server. Provided for the server side of the widely-deployed standard POP3 (RFC 1939) access protocol.

The following submodule is an executable program that is required by the protocol servers:

- Scheduler–A multithreaded session manager and resource scheduler for the protocol server daemons. This process monitors all TCP ports for which the Sun Internet Mail is providing Access Services, spawns threads for the appropriate server protocol, allocates shared resources used by multiple server sessions, and transfers ownership of resources from session to session. The scheduler also enforces administrative policy, such as the maximum number of Access sessions per server.

The following modules provide the core of the Message Access Services component:

- Internet Message Access Services Engine–A general purpose message and message store manipulation engine.
- User Properties Interface–An abstraction layer for obtaining user profile information, including authentication parameters, folder types, and quotas. Queries are passed through LDAP to the Directory Service for resolution.
- Folder Name Mapper–Parser and set of rules for mapping user-entered logical IMAP folder names to physical names. Rules are obtained from the User Properties Interface.
- Administrative Interface–Global administrative parameters for all Message Access submodules contained in flat configuration file.
- Monitoring and Statistics Interface–Query interface for returning monitoring parameters and historical statistics to the administrative console.
- OS Dependent Interface–Abstraction layer for operating system services used by the IMAS.

The internet mail access library includes a variety of different backend drivers. The drivers to be used by the application must be selected at build time. The drivers include:

- Solaris Mailbox Format Driver–This driver manipulates a Solaris Berkeley-style mailbox on a UNIX filesystem.
- Sun Message Store Driver–This driver manipulates the Sun Message Store, which is the primary Store in the Sun Internet Mail release.
- IMAP4 Client Driver–Used by mail client applications to access an IMAP server, or by an IMAP server application for proxy IMAP access. The primary use of this is to insulate users (clients) from installations where the Inbox and personal folders are contained on different servers.
- Dummy Driver–Stubs out functions not provided by all drivers. It is primarily used for mailbox name canonicalization, the file manipulation commands, and the folder commands.

Message Store

The Sun Message Store is the primary Message Store in the Sun Internet Mail release. This provides a significant advance in reliability, performance, and scalability among open systems message stores. The key features of this component are:

- Supported Internet Standards–The Message Store stores any message that conforms to RFC 822 specifications. It recognizes MIME content format and supports direct addressability of any header or body part. It is specifically designed for IMAP4 message access.
- Reliable Scalable Design–Write-once data store and two-level indexing simplify access, reduce contention, and facilitate multithreading. Committed transactions also facilitate multithreading and ensure that no messages are lost or corrupted.
- High Storage Efficiency–The Message Store retains exactly one copy of each message, regardless of the number of recipients.
- Optimized Access–Messages are preparsed and indexed when inserted into the store. No parsing is necessary when messages are accessed. The degree of preparsing is tunable. The benefits of preparsing decrease as message size increases. POP users do not need parsing at all.
- Optimized Filesystem Usage–Time-based sorting of messages within the data store provides good locality of reference and more effective use of disk caches.
- Optimized Updates–Once in the store, messages are never modified. Status changes and folder updates are stored in lightweight index files that are rapidly updated.

- Managed Backup, Migration, Archival, and Purge–Bulk dump and load facility supports backup, restore, and migration of individual users, groups, or entire stores. Deletion and purge tools support archival and guaranteed delete.

Component Architecture

The Sun Message Store is organized into the following major subcomponents:

- IMTA Delivery Queue–This is the IMTA channel queue interface that accepts messages from the IMTA and inserts them into the message store. The implementation is as follows:

  a. Messages are appended to the central data store

  b. A data store index entry is created

  c. A pointer to the index is added to the inbox folder of each recipient.
- Retrieval Interface–The retrieval interface is the c-client driver interface, used by c-client applications to manipulate messages in the Message Store. The Sun Message Store driver can be used only in protocol server applications, like the IMAP4 and POP3 servers.
- Backup and Restore Facility–This facility is the generalized bulk dump and load facility that can be used for message store backup, moving users from one message store to another.
- Administration–The Sun Message Store includes: a store-wide configuration file and user profiles obtained from the Directory.
- Monitoring–The message store provides extensive statistics, including disk space in use, number of messages in folders, oldest messages, and most recent activity. Monitoring parameters are exposed through the c-client driver interface. The set of managed objects is specified by the Message Access component.
- Maintenance Utilities–Command-line utilities are provided for periodic maintenance.

Internet Message Transfer Agent

The Internet Message Transfer Agent (IMTA) is responsible for the receipt, routing, relaying, and delivery of internet mail messages. In addition to the basic functionality of a Mail Transfer System, this component provides connections to other electronic messaging environments.

Key features of the IMTA are:

- Support for the Extended Simple Mail Transfer Protocol (ESMTP), including message size negotiation, 8/7-bit downgrading, extended error codes (RFC 1893), and full support for NOTARY Delivery Status Notifications (RFC 1891)
- Support of the LDAP Directory
- High Performance and Scalability for support of diverse messaging needs
- Support for distribution lists
- Workgroup and enterprise configurations that allow sites to control the cost and configuration complexity of the IMTA
- Support for anti-relaying (anti-spamming)
- Java-based administration

Component Architecture

The IMTA routes email messages between an arbitrary number of mail components such as the message stores, other SMTP hosts, UUCP, non-SMTP mail networks, and mail processing programs. It uses the concept of channels to accomplish this task. A channel is an interface with another mail component and is responsible for dealing with inbound and outbound email traffic between the IMTA and the mail component.

Internet email transfer is achieved by a store and forward mechanism. Each channel has a message queue for storing messages that are destined to be sent to one or more of the interfaces associated with the channel. A channel has a master program which extracts messages from the channel's queue and delivers them. It may also have a slave program to handle messages from the mail component associated with the channel.

The IMTA routes the messages by associating a destination channel with each envelop recipient. The IMTA uses a set of domain rewriting rules in its configuration file to determine the destination channel. For addresses matching the local channel, it looks up in the alias database to perform the rewriting.

The IMTA caches the directory information on all its users and distribution lists in its databases.

The job controller is an IMTA daemon responsible for scheduling message delivery. It does so by executing the channel master programs when appropriate.

Some external messaging interfaces of the IMTA:

- SMTP channel: TCP/IP-based message delivery and receipt.
- Pipe channel: used for alternative message delivery programs.
- UUCP channel: based upon periodic dial-up asynchronous communication.
- `/var/mail` channel: delivers mail to `/var/mail` for backward compatibility.

- Message store channel: delivers to the SIMS message store.
- Reprocessing channel: useful for messages that are resubmitted due to transient delivery problems.
- Defragmentation channel: reassembles partial messages into the original complete message.
- Conversion channel: performs body party-by-body part conversion on messages.

Compliance with Standards

The IMTA is fully compliant with RFC 822 and with RFCs 1521 and 1522 (MIME), the standards for the formatting of Internet text messages. SMTP support complies with RFC 821 (Simple Mail Transfer Protocol) and RFCs 1425, 1426, and 1427 (SMTP extensions). It also complies with various other Internet formats and protocols, including RFC 1623 (Internet host application requirements), RFC 1090 (SMTP on X.25), and RFC 976 (UUCP mail interchange).

See Appendix A, "Supported Standards" for a list of all of the standards supported in SIMS.

Sun Directory Services

Sun Directory Services stores information about the users, groups and services within an organization and makes this data available via *directory servers*. It provides a central repository for organizational information, and is a more manageable solution to the problem of storing and sharing organizational information than unreliable techniques that may use everything from hardcopy files, to files in singular users' home directories, or public file servers on which the data may not be accurate or kept up-to-date.

Users and applications access Directory Services using a client-server protocol called Lightweight Directory Access Protocol. (LDAP). LDAP has been widely endorsed within the Internet community and many vendors have already incorporated LDAP support into their products.

The information stored in a directory service is of little use unless it is actually being used by applications. Directory-enabled applications are essential for a Directory Service to satisfy *all* users in an organization. Web browsers, email, and calendar or scheduling applications are ideal candidates for directory enabling; others include video, telephony, and conferencing applications.

Sun LDAP Directory Service

The Sun Internet Mail product has an integrated LDAP directory. Its primary goal is to serve the needs of email users. Directory Services stores information such as email addresses, distribution lists, and the location of email servers. It also stores user information not directly related to email, such as telephone numbers, postal addresses, login IDs, and URLs. The LDAP directory can be accessed by users.

The following components provide the core functions of the LDAP Directory Service:

- Directory Server–Provides access to user, group and application data. It is a multithreaded LDAP server that uses a high-performance B-tree database for the directory data store.
- Replication Server–Controls directory data replication. It propagates directory updates using a master/slave scheme.
- Web Gateway–An HTTP/LDAP gateway that provides access to data directories from a standard World Wide Web browser.
- Directory Administration–Provided by a Java Administration Console (see the *Sun Internet Mail Server 3.5 Systems Administrator's Guide*) and several command-line tools for directory data manipulation and for database management.
- Directory Monitoring Agent–Supports directory monitoring. It provides access to directory management information.
- Directory Synchronization Tool–Converts UNIX system user and distribution-list data into LDAP directory data format.

The Sun Internet Mail product uses LDAP for directory access. However, it is not restricted to using just a single directory service. You may already have some form of directory service in use. Sun Internet Mail makes it possible to use existing data. The Sun Internet Mail product supports a native LDAP directory, making it a flexible and practical tool.

The following items define the external interfaces to the Directory Service:

- Directory Access is via the Lightweight Directory Access Protocol Version 2 (RFC 1777) and via the Hypertext Transfer Protocol (RFC 1945). LDAP data is represented according to RFC 1778, RFC 1779 and RFC 1558.
- Directory Schema describes the format and structure of directory data. The Internet/COSINE X.500 schema (RFC 1274) and ITU X.520/X.521 schemas are supported. Additional schema objects are defined to support email routing and delivery.
- External Data Representation is the LDAP Data Interchange Format (LDIF). This is a text representation of directory database data.

- Directory Monitoring is via the Simple Network Management Protocol Version 1 (RFC 1157). The X.500 Directory Monitoring MIB (RFC 1567) and the Network Services Monitoring MIB (RFC 1565) are both supported.

Administration Services

The Administration Services component contains the client and server software responsible for managing all the components of the Sun Internet Mail product. It also provides the GUI for some system functionalities. The Administration Services component is based on Java and JMAPI (Java Management API).

Key features of this component are:

- Installation
- Licensing
- Initialization and setup
- Configuration
- Maintenance
- Error recovery

Component Architecture

The Administration Services component manages:

- Directory services
- IMTA
- Sun Message Store
- Security

The following submodules provide the core functions of the Administration Services:

- Browser-based Java administration console
- Downloadable Java applets
- Java-based administration server
- HTTP server

For more information on the architecture of the Administration Services, refer to the *Sun Internet Mail 3.5 System Administrator's Guide*.

Sun Messaging Connectivity Services

Sun Messaging Connectivity Services provides batch-mode connectivity, or gateways, to the following proprietary messaging systems:

- Lotus cc:Mail
- Microsoft Mail
- IBM PROFS Mail

Sun Messaging Connectivity Services connects SIMS 3.5 to proprietary mail systems and supports integration of users from proprietary systems to native internet.

Component Architecture

SMCS is structured as a common backend (server) and a family of messaging frontends, one frontend per foreign system. Frontends provide the direct interface with the proprietary system, while the backend server is responsible for services that are common to all other connectivity services. SMCS follows the “small footprint'” model for frontends: the frontend software is small and simple, requires little administration, and is not intrusive in the proprietary message system.

Specifically, the messaging frontend has the following responsibilities:

- Use the facilities of the proprietary system to submit and extract messages in their native format.
- Use the facilities of the proprietary system to extract and update directory information.
- Move messages and directory updates in their native format between the frontend and the backend server.
- Expose the necessary administrative hooks so that the backend server can manage the frontends. It is especially important that the queuing between the frontend and backend be monitored for blockages and other problems, just as the server monitors its internal queues.

The SMCS backend is responsible for:

- N-way mapping of message formats—This can become quite complex when the originating system has capabilities missing from the target.
- Mapping recipient addresses—This is achieved by mapping native source recipient addresses into Internet domain form, then mapping back out to the destination native form. Mapping can be performed by rule (algorithmic translation), by table lookup, or a combination of rules and tables.

- Routing messages to the destination, per the recipient addresses—In the initial release, SMCS will maintain its own router component, distinct from that in the IMTA. Since all routing is done per the directory, this will be invisible to the administrator.
- Directory Synchronization—All proprietary systems maintain their own address books and user directories. SMCS is responsible for gathering these directories, merging them, resolving inconsistencies, and then exporting the master directory to the proprietary systems.
- Mapping message content—For example, MIME, Microsoft Mail, and cc:Mail use different character sets for text, and different encoding formats for bitmaps. Other systems have limits on line length, or the maximum number of characters in a text body part.

For more information on SMCS refer to the *Sun Messaging Connectivity Services cc:Mail Channel Guide*, the *Sun Messaging Connectivity Services Microsoft Mail Channel Guide*, or the *Sun Messaging Connectivity Services PROFS Channel Guide*.

APPENDIX **A**

Supported Standards

This appendix lists national, international, and industry standards related to electronic messaging and for which support is claimed by Sun Internet Mail Server 3.5. Most of these are Internet standards, published by the Internet Engineering Task Force (IETF) and approved by the Internet Activities Board (IAB). Standards documents from other sources are noted.

Several of the documents are listed with an obsolete status. These are included because they describe protocol features that were obsolete or replaced by later documents, but are still in widespread use.

Messaging

The following list of documents are relevant to national and international standards for messaging, specifically, messaging structure.

Basic Message Structure

The structure of basic messages is explained in the documents listed in TABLE A-1.

TABLE A-1 Basic Message Structure

Standard	Status	Description
RFC 822 STD 11	Standard	David H. Crocker, University of Delaware, *Standard for the Format of ARPA Internet Text Messages*, August 1982.
RFC 1123	Standard	Robert Braden (Editor), *Requirements for Internet Hosts - Application and Support*, Internet Engineering Task Force, October 1989.

Access Protocols and Message Store

The documents listed in TABLE A-2, are reference materials that contain information about access protocols and message stores.

TABLE A-2 Access Protocols and Message Store

Standard	Status	Description
RFC 1731	Proposed Standard	John G. Myers, (Carnegie-Mellon University), *IMAP4 Authentication Mechanisms*, December 1994.
RFC 1733	Information	Mark R. Crispin, (University of Washington), *Distributed Electronic Mail Models in IMAP4*, December 1994.
RFC 1939	STD 53	John G. Myers (Carnegie-Mellon University) and Marshall T. Rose (Dover Beach Consulting), *Standard Post Office Protocol - Version 3*, May 1996.
RFC 2060	Proposed Standard	Mark Crispin (University of Washington), *Internet Message Access Protocol - Version 4rev1*, December 1996.
RFC 2061	Information	Mark R. Crispin (University of Washington), *IMAP4 Compatibility With IMAP2bis*, December 1996.

SMTP and Extended SMTP

The documents listed in TABLE A-3, are reference materials that contain information about Simple Mail Transfer Protocol (SMTP) and Extended SMTP.

TABLE A-3 SMTP and Extended SMTP

Standard	Status	Description
RFC 821 STD 10	Standard	Jonathan B. Postel, USC / Information Sciences Institute, *Simple Mail Transfer Protocol*, August 1982.
RFC 1047	Information	Craig Partridge, CIC BBN Laboratories Inc., *Duplicate Messages and SMTP*, February 1988.
RFC 1428	Information	Greg Vaudreuil, Corporation for National Research Initiatives, *Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME*, February 1993.
RFC 1652	Draft Standard	John Klensin (United Nations University), Einar Stefferud (Network Management Associates, Inc.), Ned Freed (Innosoft), Marshall Rose (Dover Beach Consulting), David Crocker (Brandenburg Consulting), *SMTP Service Extension for 8bit-MIME transport*, July 1994.

TABLE A-3 SMTP and Extended SMTP

Standard	Status	Description
RFC 1869 STD 10	Standard	John Klensin (United Nations University), Ned Freed (Innosoft), Marshall Rose (Dover Beach Consulting), Einar Stefferud (Network Management Associates, Inc.), David Crocker (The Branch Office), *SMTP Service Extensions*, November 1995.
RFC 1870 STD 10	Standard	John Klensin (United Nations University), Ned Freed (Innosoft), Keith Moore (University of Tennessee), *SMTP Service Extension for Message Size Declaration*, November 1995.
RFC 1893	Proposed Standard	Greg Vaudreuil (Corporation for National Research Initiatives), *Enhanced Mail System Status Codes*, January 15, 1996.
RFC 1985	Proposed Standard	J. De Winter, *SMTP Service Extension for Remote Message Queue Starting*, August 1996.

Message Content and Structure

The following documents specify message contents handling, most of which is covered by the Multipurpose Internet Mail Extensions (MIME). There are also several non-standard message content RFCs that are supported by the SIMS product, which are listed separately, in TABLE A-4.

TABLE A-4 Message Content and Structure

Standard	Status	Description
RFC 1341	Obsolete	Nathaniel Borenstein (Bellcore) and Ned Freed (Innosoft), *MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, June 1992.
RFC 1524	Information	Nathaniel Borenstein (Bellcore), *A User Agent Configuration Mechanism For Multimedia Mail Format Information*, September 1993.
RFC 1806	Experimental	Rens Troost (New Century Systems), Steve Dorner (Qualcomm), *Communicating Presentation Information in Internet Messages: The Content-Disposition Header*, June 1995.
RFC 2017	Proposed Standard	Ned Freed (Innosoft), Keith Moore (University of Tennessee), *Definition of the URL MIME External-Body Access-Type*, October 1996.

TABLE A-4 Message Content and Structure

Standard	Status	Description
RFC 2045	Draft Standard	Nathaniel Borenstein (First Virtual Holdings) and Ned Freed (Innosoft), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, November 1996.
RFC 2046	Draft Standard	Nathaniel Borenstein (First Virtual Holdings) and Ned Freed (Innosoft), *MIME Part Two: Media Types*, November 1996.
RFC 2047	Draft Standard	Keith Moore (University of Tennessee), *MIME Part Three: Message Header Extensions for Non-ASCII Text*, November 1996.
RFC 2048	Policy	Ned Freed (Innosoft), John Klensin (MCI), Jon Postel (USC / Information Sciences Institute), *MIME Part Four: Registration Procedures*, November 1996.
RFC 2049	Draft Standard	Nathaniel Borenstein (First Virtual Holdings) and Ned Freed (Innosoft), *MIME Part Five: Conformance Criteria and Examples*, November 1996.

Delivery Status Notifications

The list of documents in TABLE A-5 describe delivery status notification.

TABLE A-5 Delivery Status Notifications

Standard	Status	Description
RFC 1891	Proposed Standard	*SMTP Service Extension for Delivery Status Notifications*, Keith Moore (University of Tennessee), January 15, 1996.
RFC 1892	Proposed Standard	Greg Vaudreuil (Corporation for National Research Initiatives), *The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages*, January 15, 1996.
RFC-1894	Proposed Standard	Keith Moore (University of Tennessee), Greg Vaudreuil (Corporation for National Research Initiatives), *An Extensible Message Format for Delivery Status Notifications*, January 15, 1996.

Domain Name Service

The following documents listed in TABLE A-6 specify the naming facilities of the Internet and how those facilities are used in messaging.

TABLE A-6 Domain Name Service

Standard	Status	Description
RFC 920	Policy	Jonathan B. Postel and Joyce K. Reynolds, USC / Information Sciences Institute, *Domain Requirements*, October 1984.
RFC 974	Standard	Craig Partridge, CSNET CIC BBN Laboratories Inc., *Mail Routing and the Domain System*, January 1986.
RFC 1032	Information	Mary K. Stahl, SRI International, *Domain Administrators Guide*, November 1987.
RFC 1033	Information	Mark K. Lottor, SRI International, *Domain Administrators Operations Guide*, November 1987.
RFC 1034	Standard	Paul V. Mockapetris, USC / Information Sciences Institute, *Domain names - concepts and facilities*, November 1987.
RFC 1035	Standard	Paul V. Mockapetris, USC / Information Sciences Institute, *Domain names - Implementation and Specification*, November 1987.

Directory

The following list of documents are relevant to national and international standards for directory server specifications.

Server Specification

The reference material listed in TABLE A-7 describes international standards for server specifications.

TABLE A-7 Server Specification

Standard	Status	Description
ITU X.520	International Standard	ITU-T Recommendation X.520(1993), ISO / IEC 9594-6, *Information Technology - Open Systems Interconnection - The Directory: Selected Attribute Types.*
ITU X.521	International Standard	ITU-T Recommendation X.521(1993), ISO / IEC 9594-7.X, *Information Technology - Open Systems Interconnection - The Directory: Selected Object Classes.*
RFC 1274	Proposed Standard	Paul Barker and Steve Kille, University College London, *The COSINE and Internet X.500 Schema*, November 1991.
RFC 1279	Information	Steve Kille, University College London, *X.500 and Domain*s, November 1991.
RFC-1781	Proposed Standard	Steve Kille (ISODE Consortium), *Using the OSI Directory to Achieve User Friendly Naming*, March 1995.
RFC 1801	Experimental	Steve Kille (ISODE Consortium), *MHS use of the X.500 Directory to support MHS Routing*, June 1995.
RFC 1803	Information	Russ Wright (Lawrence Berkeley Laboratory), Arlene F. Getchell (Lawrence Livermore National Laboratory), Tim Howes (University of Michigan), Srinivas R. Sataluri (AT&T Bell Laboratories), Peter Yee (Ames Research Center), and Wengyik Yeong (PSI, Inc.), *Recommendations for an X.500 Production Directory Service*, June 1995.

Access Protocols

The reference material listed in TABLE A-8 describes information about access protocols.

TABLE A-8 Access Protocols

Standard	Description
RFC-1777	Wengyik Yeong (PSI, Inc.), Tim Howes (University of Michigan), and Steve Kille (ISODE Consortium), *Lightweight Directory Access Protocol*, March 1995.
RFC-1778	Tim Howes (University of Michigan), Steve Kille (ISODE Consortium), Wengyik Yeong (PSI, Inc.), and Colin Robbins (NeXor Ltd), *The String Representation of Standard Attribute Syntaxes*, March 1995.
RFC-1779	Steve Kille (ISODE Consortium), *A String Representation of Distinguished Names*, March 1995.
RFC-1798	Alan Young (ISODE Consortium), *Connection-less Lightweight Directory Access Protocol*, June 1995.

Text and Character Set Specifications

The following tables contain reference material that describes national and international telecommunications and information processing requirements.

National and International

TABLE A-9 contains a list of reference material pertaining to national and international telecommunications and information exchange standards.

TABLE A-9 National and International Information Exchange

Standard	Status	Description
IA5	International Standard	ITU-T Recommendation T.50, Fascicle VII.3, Malaga-Torremolinos, *International Alphabet No. 5, International Telecommunication Union*, 1984, Geneva, 1989.
ISO 2022	International Standard	International Organization for Standardization (ISO), *Information processing - ISO 7-bit and 8-bit coded character sets - Code extension techniques*, Ref. No. ISO 2022-1986.
JIS X 0201	National Standard	Japanese Standards Association, *Code For Information Interchange*, JIS X 0201-1976.
JIS X 0208	National Standard	Japanese Standards Association, *Code of the Japanese Graphic Character Set For Information Interchange*, JIS X 0208-1990.
JUNET	Public Network	JUNET Riyou No Tebiki Sakusei Iin Kai (JUNET User's Guide Drafting Committee), *JUNET Riyou No Tebiki (JUNET User's Guide)*, First Edition, February 1988.
US ASCII	National Standard	American National Standards Institute, ANSI X3.4-1986, *Coded Character Set-7-bit American national standard code for information interchange*. New York, 1986.
US LATIN	National Standard	American National Standards Institute, ANSI Z39.47-1985, *Coded Character Set-Extended Latin alphabet code for bibliographic use*. New York, 1985.

Internet

The reference material in TABLE A-10 lists available documentation that describes Internet communications and standards.

TABLE A-10 Internet

Standard	Status	Description
RFC 1345	Information	Keld Simonsen, Rationel Almen Planlaegning, Internet Activities Board RFC 1345, *Character Mnemonics & Character Sets*, June 1992.
RFC-1468	Information	Jun Murai (Keio University), Mark Crispin (University of Washington), *Japanese Character Encoding for Internet Messages*, June 1993.
RFC-1502	Information	Harald Tveit Alvestrand, SINTEF DELAB, Internet Activities Board RFC 1502, *X.400 Use of Extended Character Sets*, August 1993.

Glossary

abstract syntax	A description of a data structure that is independent of machine-oriented structures and encoding.
ACAP	A protocol which enhances IMAP by allowing the user to set up address books, user options, and other data for universal access.
access control rules	Rules that define which users are granted which permissions for a given set of directory entries or attributes.
ACSE	Association Control Service Element. The method used in OSI for establishing a call between two applications. Checks the identities and contexts of the application entities, and could apply an authentication security check.
Administration Console or Admin Console	A GUI (graphical user interface) which enables you to configure, monitor, maintain, and troubleshoot the SIMS components.
address mapping	See forward address mapping or reverse address mapping.
address resolution	A means for mapping Network Layer addresses onto media-specific addresses. See also *ARP*.
address token	The address element of a rewrite rule pattern.
ADMD	Administration Management Domain. An X.400 Message Handling System public service carrier. Examples: MCImail and ATTmail in the U.S., British Telecom Gold400mail in the U.K. The ADMDs in all countries worldwide together provide the X.400 backbone. See also *PRMD*.
Administration Services	Administers all components of SIMS through a JMAPI-based GUI. See also *JMAPI*.
agent	In the client-server model, the part of the system that performs information preparation and exchange on behalf of a client or server application. See also *NMS, DUA, MTA*.
alias	An address which delivers messages to a specified group of users. A listserve.

alias entry	Contains the name of the directory entry it represents in the directory information tree, and can also contain other attributes. It is identified by the distinguished name.
alias file	A file used to set aliases not set in a directory, such as the postmaster alias.
ANSI	American National Standards Institute. The U.S. standardization body. ANSI is a member of the International Organization for Standardization (ISO).
API	Application Program Interface. A set of calling conventions defining how a service is invoked through a software package.
Application Layer	The top-most layer in the OSI Reference Model providing such communication services as electronic mail and file transfer.
ASM	Application Specific Module. An example of this is an external Solaris Backup.
ASN.1	Abstract Syntax Notation One. The OSI language for describing abstract syntax. See also *BER*.
attribute	The form of information items provided by the Directory Service. The directory information base consists of entries, each containing one or more attributes. Each attribute consists of a type identifier together with one or more values. Each directory Read operation can retrieve some or all attributes from a designated entry.
attribute index	An index, or list, of entries which contains a given attribute or attribute value.
autoreply option file	A file used for setting options for autoreply, such as vacation notices
AVM	Admin View Module. An extension of Java's Abstract Window Toolkit that provides the Administration Console's graphical user interface.
AWT	Abstract Window Toolkit. A Java development toolkit.
backbone	The primary connectivity mechanism of a hierarchical distributed system. All systems that have connectivity to an intermediate system on the backbone are connected to each other. This does not prevent you from setting up systems to bypass the backbone for reasons of cost, performance, or security.
backend	Stores directory information. There are two types of backends; LDBM which provides access to information stored in a database, and shell which provides access to information stored in any format.
backup	The process of dumping the contents of folders from the Sun Message Store to a backup device. See also *purge* and *restore*.
BER	Basic Encoding Rules. Standard rules for encoding data units described in ASN.1. Sometimes incorrectly lumped under the term ASN.1, which properly refers only to the abstract syntax description language, not the encoding technique.

big-endian	A format for storage or transmission of binary data in which the most significant bit (or byte) comes first. The reverse convention is called little-endian.
BOC	Bell Operating Company. More commonly referred to as RBOC for Regional Bell Operating Company. The local telephone company in each of the seven U.S. regions.
CA	Certificate Authority. An organization that issues digital certificates (digital identification) and makes its public key widely available to its intended audience.
cache	A temporary storage file of information that has been retrieved from the directory.
CCITT	See also *ITU*.
chaining	The directory server passes an information request to the Directory Service Agent (DSA) that can process the request. The second DSA returns the result to the first DSA, which then returns it to the client. See also *knowledge information*.
channel	An interface with another Sun Internet Mail Server version component, another email system, or a mail user agent.
character set labels	SIMS can be configured to process either 7 or 8 bit character sets, by using the menus in the Channel Property book. This configuration will affect encrypted and possibly garbled messages received from other systems. For more detailed instructions, see pg. 124 of the Adminstator's Guide.
ciphertext	Data that has been coded (enciphered, encrypted, or encoded) for security purposes.
client-server model	A common way to describe network services and the model user processes (programs) of those services. Examples include the name-server / name-resolver paradigm of the DNS and fileserver / file-client relationships such as NFS and diskless hosts.
composition	The process of constructing a message by the Mail User Agent (MUA). See also *MUA*.
congestion thresholds	A limit on disk space set by the system administrator which prevents the database from becoming overloaded by restricting new operations when system resources are insufficient.
content	The content of a message provides the data that the originator of the message intends to transmit to the recipient.The content of a message can contain text as well as images, audio, video, and binary or application-specific files.
content-transfer encoding	Specifies how data is encoded so the data can traverse Internet Mail Transport Agents (IMTAs) outside of the SIMS email system that may have data or character-set limitations.

conversion channel	Converts body of messages from one form to another.
cross reference	Any naming context that can be contacted directly. See also *knowledge information*.
data store	A store that contains directory information, typically for an entire directory information tree.
DCE	Distributed Computing Environment. An architecture of standard programming interfaces, conventions, and server functionalities (e.g., naming, distributed file system, remote procedure call) for distributing applications transparently across networks of heterogeneous computers. Promoted and controlled by the Open Software Foundation (OSF), a consortium led by HP, DEC, and IBM. See also *ONC*.
defragmentation	The Multiple Internet Extensions (MIME) feature that enables a large message that has been broken down into smaller messages or fragments to be reassembled. A Message Partial Content-Type header field that appears in each of the fragments contains information that helps reassemble the fragments into one message. See also *fragmentation*.
denial of service attack	A situation where an individual intentionally or inadvertently overwhelms your mail server by flooding it with messages. Your server's throughput could be significantly impacted or the server itself could become overloaded and nonfunctional.
Departmental Edition	Also referred to as Sun Internet Mail Server-Departmental Edition, is the version of SIMS intended for use by local departmental environments. This package performs its own routing and delivery within a local office or department, but hands off interdepartmental mail to backbone or enterprise server.
dereferencing an alias	Specifying, in a bind or search operation, that a directory translate an alias DN to the DN of an actual entry.
destination channel	The last element of a host/domain rewrite rule, in whose queue a message should be placed in for delivery.
directory context	The point in the directory tree at which a search is begun.
directory entry	A set of directory attributes and their values identified by its distinguished name. Each entry contains an object class attribute that specifies the kind of object the entry describes and defines the set of attributes it contains.
Directory Information Tree	Directory Information Tree is the tree-like hierarchical structure in which directory entries are organized.
directory schema	The set of rules that defines the data that can be stored in the directory.

Directory Service	A logically centralized repository of information. The component in SIMS that stores user, distribution list, and configuration data.
dirsync option file	A file used to set options for the dirsync program which cannot be set through the command line.
disconnected state	The mail client connects to the server, makes a cache copy of selected messages, then disconnects from the server.
distinguished name	The sequence of attributes and values of an entry within the directory information tree.
distribution list	A list of email addresses (users) that can be sent a message by specifying one email address. See also *expansion*, *member, moderator, owner,* and *alias*.
DIT	Directory information tree. A hierarchical structure in which directory data or information (names, email addresses, and so on) is stored.
DNS	Domain Name Service. The naming facilities of the Internet.
domain	In the Internet, a part of a naming hierarchy. Syntactically, an Internet domain name consists of a sequence of names (labels) separated by periods (dots), for example, `tundra.mpk.ca.us`. In OSI, `domain` is generally used as an administrative partition of a complex distributed system, as in MHS Private Management Domain (PRMD), and Directory Management Domain (DMD).
domain rewriting rules	See also *rewrite rules*.
domain template	The part of a rewrite rule that defines how the host/domain portion of an address is rewritten. It can include either a full static host/domain address or a single field substitution string, or both.
dotted decimal notation	The syntactic representation for a 32-bit integer that consists of four 8-bit numbers written in base 10 with periods (dots) separating them. Used to represent IP addresses in the Internet as in 192.67.67.20.
DSA	Directory System Agent. The software that provides the X.500 Directory Service for a portion of the directory information base. Generally, each DSA is responsible for the directory information for a single organization or organizational unit.
DUA	Directory User Agent. The software that accesses the X.500 Directory Service on behalf of the directory user. The directory user may be a person or another software element.
EMAPI	Extended MAPI Service Provider. Transparently turns Microsoft Exchange client into an Internet standard IMAP/LDAP client. See also *IMAP, LDAP*.
encapsulation	The technique used by layered protocols in which a layer adds header information to the protocol data unit (PDU) from the layer above.

encryption	Scrambling the contents of a message so that its contents cannot be read without the encryption, or code key.
end system	An OSI system which contains application processes capable of communicating through all seven layers of OSI protocols. Equivalent to Internet host.
Enterprise Edition	Also referred to as Sun Internet Mail Server-Enterprise Edition, provides full-featured messaging server for large user communities.
entity	OSI terminology for a layer protocol machine. An entity within a layer performs the functions of the layer within a single computer system, accessing the layer entity below and providing services to the layer entity above at local service access points.
entries	User, group, or organizational data used to configure message accounts.
envelope	The part of an Internet mail message that contains the delivery information. The envelope contains the originator and recipient information associated with a message.
ESMTP	Extended Simple Mail Transfer Protocol. An Internet message transport.
expander	Part of an electronic mail delivery system which allows a message to be delivered to a list of addressees. Mail exploders are used to implement mailing lists. Users send messages to a single address (e.g., hacks@somehost.edu) and the mail exploder takes care of delivery to the individual mailboxes in the list.
expansion	This term applies to the Internet Mail Transport Agent (IMTA) processing of distribution lists. The act of converting a message addressed to a distribution list into enough copies for each distribution list member.
expunge	The act of deleting a message then removing the deleted message via a mail client.
external channel	An interface between the IMTA and either another SIMS component or another component outside the SIMS email system.
File System	This can be either safe or unsafe. A safe file system performs logging such that if a system crashes it is possible to rollback the data to a pre-crash state and restore all data. An example of a safe file system is VXFS. An unsafe file system does not perform logging. If the system crashes, the state cannot be recreated and some data may be lost. You must also perform imcheck before activating message access to these files.
firewall	Router or mail-level hosts that are equipped with special codes to control access between the Internet and the internal network.
folder	Named place where mail is stored. Also called a *mailbox*. Inbox is a folder that stores new mail. Users can also have folders where mail can be stored. A folder can contain other folders in a hierarchical tree. Folders owned by a user are called *private folders*. See also *shared folders*.

Folder Check	A utility which checks the accessibility of messages and folders and verifies links. This utility is used as part of the regular maintenance of SIMS.
forward address mapping	Message envelopes, TO:address, are processed to a mapping table. The result of the mapping is tested. If necessary, the exact form of the envelope is exchanged for another which can then be processed by a different, and perhaps non-compliant RFC 822, mail system.
fragmentation	The Multiple Internet Extensions (MIME) feature that allows the breaking up of a large message into smaller messages. See also *defragmentation*.
FTAM	File Transfer, Access, and Management. The OSI remote file service and protocol. See also *FTP*.
FTP	File Transfer Protocol. The Internet protocol (and program) used to transfer files between hosts. See also *FTAM*.
full static host/domain address	The portion of a host/domain address elements set off by decimals as part of the domain template. See also *domain template*.
gateway	The terms *gateway* and *application gateway* refer to systems that do translation from one native format to another. Examples include X.400 to/from RFC 822 electronic mail gateways. A machine that connects two or more electronic mail systems (especially dissimilar mail systems on two different networks) and transfers messages between them. Sometimes the mapping and translation can be complex, and it generally requires a store-and-forward scheme whereby the message is received from one system completely before it is transmitted to the next system after suitable translations.
global log manager	A utility that handles log information from each Sun Internet Mail Server component.
GOSIP	Government OSI Profile. A U.S. Government procurement specification for OSI protocols.
group entry	See distribution lists.
group folders	Contain folders for shared and group folders.
header	The part of an Internet mail message that is composed of a field name followed by a colon and then a value. Headers include delivery information, summaries of contents, tracing, and MIME information.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
IAB	Internet Activities Board. The technical body that oversees the development of the Internet suite of protocols (commonly referred to as “TCP/IP”). It has two task forces (the IRTF and the IETF) each charged with investigating a particular area.

IESG Internet Engineering Steering Group	The executive committee of the IETF.
IETF Internet Engineering Task Force	One of the task forces of the IAB. The IETF is responsible for solving short-term engineering needs of the Internet. It has over 40 Working Groups.
IMAP4	Internet Message Access Protocol. IMAP4 provides advanced disconnected mode client access.
IMS log	Sun Message Store log files.
IMTA	Internet Message Transfer Agent. IMTA routes, transports, and delivers Internet Mail messages within the email system.
intermediate system	An OSI system which is not an end system, but which serves instead to relay communications between end systems.
internal channel	An interface between internal modules of the IMTA. Internal channels include the reprocessing, conversion, and defragmentation channels. These channels are not configurable.
Internet address	A 32-bit address assigned to hosts using TCP/IP. See also *dotted decimal notation*.
IP Internet Protocol	The network layer protocol for the Internet protocol suite.
ISDN	Integrated Services Digital Network. An emerging technology which is beginning to be offered by the telephone carriers of the world. ISDN combines voice and digital network services in a single medium making it possible to offer customers digital data services as well as voice connections through a single “wire.” The standards that define ISDN are specified by ITU-T.
ISO	International Organization for Standardization. See also *OSI*.
ITU	International Telecommunications Union. International Consultative Committee for Telegraphy and Telephony. A unit of the International Telecommunications Union (ITU) of the United Nations. An organization with representatives from the PTTs of the world. CCITT produces technical standards, known as “Recommendations,” for all internationally controlled aspects of analog and digital communications. See also *X Recommendations*.
Java	A programming language developed by Sun Microsystems.
JMAPI	Java Management Application Programming Interface. JMAPI is a collection of programming language classes that enable a diverse set of autonomous applications to be brought together under a common look, feel, and behavior everywhere they run.

job controller	The SIMS component which schedules message delivery or message submission tasks between various SIMS components. Job controller also controls channel queues and determines the order of processing. Requests are processed in the order in which they are received by the system.
Kerberos	Client-to-server security package produced by MIT.
key ring	A collection of public and private security keys.
knowledge information	Part of the directory service infrastructure information. The directory server uses knowledge information to pass requests for information to other servers.
LDAP	Lightweight Directory Access Protocol. LDAP is a protocol used for the storage, retrieval, and distribution of information, including user profiles, distribution lists, and configuration data.
LDAP filter	A way of specifying a set of entries, based on the presence of a particular attribute or attribute value.
LDBM	A type of backend that stores directory information that provides access to information stored in a database. See also *backend*.
LDIF	LDAP Data Interchange Format. A data format used to represent LDAP entries in text form.
Legacy Mail Services	Provides batch-mode connectivity to legacy proprietary message transfer systems.
little-endian	A format for storage or transmission of binary data in which the least significant byte (bit) comes first. See also *big-endian*.
local channel	A channel that allows you to determine delivery options of local users and delivers mail to Solaris Operating Environment mailboxes.
lookup	Same as a search, using the specified parameters for sorting data.
Mailbox	A place where messages are stored and viewed. See *folder*.
Mailtool	A `/var/mail` client application that runs under the OpenWindows V3 desktop environment.
man page	UNIX Reference manual pages.
managed object	A collection of configurable attributes, for example, a collection of attributes for the directory service.
mapping tables	Two column tables which transform, map, an input string into an output string.
master directory server	The directory server that contains the data that will be replicated.

master message catalog	Contains message catalogs for the SIMS components.
master program	A channel program that initiates a message transfer to another interface on its own.
member	A user or group who receives a copy of an email addressed to a distribution list. See also *distribution list*, *expansion*, *moderator*, and *owner*.
Message Access and Store	These are the SIMS components which store user messages and allow for retrieval and processing of messages.
Message Access Services	Consists of protocol servers, software drivers, and libraries which support client access to the message store.
message catalogs	The log messages, command line responses, and graphical user interface screen text contained in the SIMS components.
message databases	Contain messages and attachments.
message hash	Contain hashing files.
message indices	Contain message index files.
message submission	The client Mail User Agent (MUA) transfers a message to the mail server and requests delivery.
MHS Message Handling System	The system of message user agents, message transfer agents, message stores, and access units which together provide OSI electronic mail. MHS is specified in the ITU-T X.400 series of Recommendations.
MIB	Management Information Base. A collection of objects that can be accessed via a network management protocol. See also *SMI*.
MIME	Multipurpose Internet Mail Extensions. A format for defining email message content.
moderator	If the moderator feature is enabled, a message addressed to a distribution list is initially sent to the moderator only. The moderator can take one of the following actions: forward the message to the distribution list, edit the message and then forward it to the distribution list, or not forward the message to the distribution list. See also *distribution list*, *expansion*, *member*, and *owner*.
MTA	Message Transfer Agent. An OSI application process used to store and forward messages in the X.400 Message Handling System. Equivalent to Internet mail agent.
MUA	Mail User Agent. The client applications invoked by end users to read, submit, and organize their electronic mail.

multicasting	A process by which the directory server broadcasts an information request to all DSAs it knows about. Any DSA that can process the request does so and returns the result to the first DSA. See also *knowledge information*.
multithreaded	The ability to handle multiple, simultaneous sessions in a single process.
name resolution	The process of mapping a name into the corresponding address. See also *DNS*.
naming attribute	The final attribute in a directory information tree distinguished name. See also *relative distinguished name*.
naming context	A specific subtree of a directory information tree that is identified by its DN. In SIMS, specific types of directory information are stored in naming contexts. For example, a naming context which stores all entries for marketing employees in the XYZ Corporation at the Boston office might be called ou=mktg, ou=Boston, o=XYZ, c=US.
network address	See also *Internet address* or *OSI Network Address*.
Network Layer	The OSI layer that is responsible for routing, switching, and subnetwork access across the entire OSI environment.
NFS®	Network File System. A distributed file system developed by Sun Microsystems which allows a set of computers to cooperatively access each other's files in a transparent manner.
NIC	Network Information Center. Originally there was only one, located at SRI International and tasked to serve the ARPANET (and later DDN) community. Today, there are many NICs, operated by local, regional, and national networks all over the world. Such centers provide user assistance, document service, training, and much more.
NIST	National Institute of Standards and Technology. (Formerly NBS, National Bureau of Standards). See also *OIW*.
NMS Network Management Station	The system responsible for managing a (portion of a) network. The NMS talks to network management agents, which reside in the managed nodes, via a network management protocol. See also *agent*.
nondelivery report	During message transmission, if the IMTA does not find a match between the address pattern and a rewrite rule, the IMTA sends a nondelivery report back to the sender with the original message, then deletes its copy of the message.
non-specific subordinate reference	A naming context that is lower in the directory tree but not a child of the naming context held by your directory server. See also *knowledge information*.
Notary Messages	Text messages sent by the MTA to an email sender indicating delivery or non-delivery status of a sent message.

object class	These are divided into 3 classes of attributes: required, reserved, or optional. Each object supports a number of various attributes. Required attributes are essential to the functionality of SIMS. Reserved attributes are for the future use of SIMS and should not be used. Optional attributes are not used, nor have any planned use, by SIMS.
off-line state	The mail client fetches messages from a server system to a client system, which may be a desktop or portable system and may delete them from the server. The mail client downloads the messages where they can be viewed and answered.
on-line state	A state in which messages remain on the server and are remotely responded to by the mail client.
ONCTM Open Network Computing	A distributed applications architecture promoted and controlled by a consortium led by Sun Microsystems.
option files	IMTA option files contain global parameters used to override default values of parameters which apply to IMTA as a whole, such as sizes for various tables into which various configuration and alias files are read.
organizational unit	A layer in the directory information tree.
OSI Open Systems Interconnection	An international standardization program to facilitate communications among computers from different manufacturers. See also *ISO*.
OSI Network Address	The address, consisting of up to 20 octets, used to locate an OSI Transport entity. The address is formatted into an Initial Domain Part which is standardized for each of several addressing domains, and a Domain Specific Part which is the responsibility of the addressing authority for that domain.
owner	An individual who is responsible for a distribution list. An owner can add or delete distribution list members. See also *distribution list*, *expansion*, *member*, and *moderator*.
PGP	Pretty Good Privacy. PGP provides client-to-client security, encrypting or scrambling the text of a message so that only the receiving message server can decrypt or unscramble the text.
permanent failure	An error condition that occurs during message handling. When this occurs, the message store deletes its copy of an email message. The Internet Message Transport Agent (IMTA) bounces the message back to the sender and deletes its copy of the message.
pipe channel	A channel which performs delivery of messages via a per-user-site-supplied program. These programs must be registered in SIMS by the system administrator, and thus do not pose a security risk.
POP	Post Office Protocol. POP provides remote access support for older mail clients.

populating the directory	Entering information for users and distribution lists to the SIMS configuration files.
PPP	Point-to-Point Protocol. The successor to SLIP, PPP provides router-to-router and host-to-network connections over both synchronous and asynchronous circuits. See also *SLIP*.
PRMD	Private Management Domain. An X.400 Message Handling System private organization mail system. Example NASAmail. See also *ADMD*.
protocol	A formal description of messages to be exchanged and rules to be followed for two or more systems to exchange information.
proxy	The mechanism whereby one system “fronts for” another system in responding to protocol requests. Proxy systems are used in network management to avoid having to implement full protocol stacks in simple devices, such as modems.
public key encryption	A cryptographic method that uses a two-part key (code) that is made up of public and private components. To encrypt messages, the published public keys of the recipients are used. To decrypt the messages, the recipients use their unpublished private keys known only to them.
purge	The process of removing messages that are no longer referenced in user and group folders and returning the space to the Sun Message Store file system. See also *backup* and *restore*.
PTT	Post Telephone Telegraph.
Qualcomm Eudora	A mail client produced by Qualcomm Corporation that supports MIME, POP3, and MAIL protocols.
quota	See user quota.
RBOC	Regional Bell Operating Company. See also *BOC*.
referral	A process by which the directory server returns an information request to the client that submitted it, with information about the Directory Service Agent (DSA) that the client should contact with the request. See also *knowledge information*.
relaying	A message is passed from one mail server to another mail server.
relative distinguished name	The final attribute and its value in the attribute and value sequence of the distinguished name. See also *distinguished name*.
replica directory server	The directory that will receive a copy of all or part of the data.
reprocessing channel	Performs deferred processing. The reprocessing channel is the intersection of all other channel programs. It performs only the operations that are shared with other channels.

restore	The process of restoring the contents of folders from a backup device to the Sun Message Store. See also *backup* and *purge*.
reverse address mapping	Addresses are processed to a mapping table, with a reversal database, generally substituting a generic address, possibly on a central machine, for an address on a remote or transitory system.
rewrite rules	Also known as domain rewriting rules. A tool that the Internet Mail Transport Agent (IMTA) uses to route messages to the correct host for delivery. Rewrite rules perform the following functions: (1) extract the host/domain specification from an address of an incoming message, (2) match the host/domain specification with a rewrite rule pattern, (3) rewrite the host/domain specification based on the domain template, and (4) decide which IMTA channel queue the message should be placed in.
RFC	Request For Comments. The document series, begun in 1969, describes the Internet suite of protocols and related experiments. Not all (in fact very few) RFCs describe Internet standards, but all Internet standards are published as RFCs. See http://www.imc.org/rfcs.html.
RMI	Remote Method Invocation. A Java-based programming language that enables the Administration Console and the server to communicate.
Roam	The Sun Internet Mail client. Roam is a disconnected mode mail user agent (MUA) that supports the low-bandwidth IMAP protocol extensions of the Sun Internet Mail Server IMAP server.
root entry	The first entry of the directory information tree (DIT) hierarchy.
router	A system responsible for determining which of several paths network traffic will follow. It uses a routing protocol to gain information about the network, and algorithms to choose the best route based on several criteria known as "routing metrics." In OSI terminology, a router is a Network Layer intermediate system. See also *gateway*.
routability scope	Specifications which enable the IMTA to send messages by the most direct route, either to a specific user's folder, a group of folders, or to a mail host.
routing	In an email system, the act of delivering a message based on addressing information extracted from the body of the message. The Internet Message Transfer Agent (IMTA) is the component responsible for routing messages.
RTSE	Reliable Transfer Service Element. A lightweight OSI application service used above X.25 networks to handshake application PDUs across the Session Service and TP0. Not needed with TP4, and not recommended for use in the U.S. except when talking to X.400 ADMDs.
S/Key	Client-to-server security package produced by Bell Labs
SASL	Server-to-server security.

schema	A set of rules which sets the parameters of the data stored in a directory. It defines the type of entries, their structure and their syntax.
shared folder	*A* mailbox that can be viewed by members of a *distribution list*. Shared folders have an *owner* who can add or delete members to the group and can delete messages from a the shared folder. The can also have a moderator who can edit, block, or forward incoming messages.
shell backend	A type of backend that stores directory information. This type provides access to information stored in any format, using shell scripts.
single field substitution string	Part of the domain template that dynamically rewrites the specified address token of the host/domain address. See also *domain template.*
SIMS Host	Name of host on which Sun Message Store is installed.
SIMS initialization duration in days	Number of days to initialize the Sun Message Store
SIMS Owner	Person in control of Sun Message Store files.
SKIP	Simple Key management for IP. A security system that encrypts or scrambles the text of a message so only the receiving mail client or message server can decrypt or unscramble the text.
SLAPD	A daemon that operates that accesses the database files that hold the directory information, and communicates with directory clients using the LDAP protocol.
slave program	A channel program that accepts transfers initiated by another interface.
SLIP	Serial Line IP. An Internet protocol used to run IP over serial lines such as telephone circuits or RS-232 cables interconnecting two systems. SLIP is now being replaced by PPP. See also *PPP.*
SLURPD	A replication daemon that runs on demand or schedule and ensures that any directory information changes are propagated to systems that hold replicas of that information.
smart host	The Internet Message Transfer Agent (IMTA) in a particular domain to which other IMTAs acting as routers forward messages if they do not recognize the recipients.
SMI	Structure of Management Information. The rules used to define the objects that can be accessed via a network management protocol. See also *MIB.*
SMTP	Simple Mail Transfer Protocol. The Internet electronic mail protocol. Defined in RFC 821, with associated message format descriptions in RFC 822.

SMPT Dispatcher	A multithreaded connection dispatching agent which allows multiple multithreaded servers to share responsibility for a given service, thus allowing several multithreaded SMTP servers to run concurrently and handle one or more active connections.
SMTP intranet or internet channel	A channel dedicated to relaying messages between the IMTA and a group of SMTP hosts within, or outside of, your mail network.
SMTP router channel	SMTP channel that handles messages between the IMTA and firewall host.
SNMP	Simple Network Management Protocol. The network management protocol of choice for TCP/IP-based internets.
Sun Directory Services	SIMS component which provides access and maintenance of user profiles, distribution lists, and other system resources.
Sun Internet Mail Client	The client end of the SIMS solution that supports online, offline, and disconnected states.
Sun Internet Mail Server	An enterprise-wide, open-standards based, scalable electronic message-handling system.
Sun Messaging Connectivity Services	This component of SIMS provides batch-mode connectivity to proprietary message transfer systems, including: “LAN mail” systems, Lotus cc:Mail, Microsoft Mail and mainframe-based IBM OfficeVision.
Sun Message Store	The server from which mail clients retrieve and submit messages.
SQL	Structured Query Language. The international standard language for defining and accessing relational databases.
subordinate reference	The naming context that is a child of the naming context held by your directory server. See also *knowledge information*.
Sun Directory Services	SIMS component which provides access and maintenance of user profiles, distribution lists, and other system resources.
Sun Messaging Connectivity Services	This component of SIMS provides batch-mode connectivity to proprietary message transfer systems, including: “LAN mail” systems, Lotus cc:Mail, Microsoft Mail and mainframe-based IBM OfficeVision.
SSL	Secure Sockets Layer is an open, non-proprietary security protocol.
synchronization	The update of data by a master directory server to a replica directory server.

table lookup	With a table consisting of two columns of data, an input string is compared with the data within the table and transformed to an output string.
tailor file	An option file used to set the location of various IMTA components.
TCP	Transmission Control Protocol. The major transport protocol in the Internet suite of protocols providing reliable, connection-oriented, full-duplex streams. Uses IP for delivery. See also *TP4*.
transient failure	An error condition that occurs during message handling. The remote Internet Message Transport Agent (IMTA) is unable to handle the message when it's delivered, but may be able to later. The local IMTA returns the message to the channel queue and schedules it for retransmission at a later time.
transport protocols	Provides the means to transfer messages between message stores.
UA	User Agent. An OSI application process that represents a human user or organization in the X.400 Message Handling System. Creates, submits, and takes delivery of messages on the user's behalf.
user entry or user profile	Fields that describe information about each user, required and optional, examples are: distinguished name, full name, title, telephone number, pager number, login name, password, home directory, etc.
user folders	Contain user's email folders.
user quota	The finite amount of space, configured by the system administrator, allocated to each user for incoming or stored messages.
user redirection	The remote Internet Message Transport Agent (IMTA) cannot accept mail for the recipient, but can reroute the mail to a mail server that can accept it.
upper reference	Indicates the directory server that holds the naming context above your directory server's naming context in the directory information tree (DIT).
UUCP	UNIX to UNIX Copy Program. A protocol used for communication between consenting UNIX systems.
UUCP Channel	Unix to Unix Copy System is provided only in the SIMS Enterprise Edition. It is a asynchronous terminal line-based system used to provide support for file transfer and remote execution between different computer systems.
`/var/mail`	The UNIX version 7 "`From`" delimited mailbox as implemented in the Solaris operating system.
workgroup	Local workgroup environment, where the server performs its own routing and delivery within a local office or workgroup. Interdepartmental mail is routed to a backbone server. See also *backbone*.
X.400	A message handling system standard.
XFN	Federated Naming Technology.

X Recommendations	The CCITT documents that describe data communication network standards. Well-known ones include X.25 Packet Switching standard, X.400 Message Handling System, and X.500 Directory Services.

Index