Sun Internet Mail Server™ 4.0 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.: 805-7677-10
Revision A, July 1999

Contents

Figures

Tables

Preface

Sun™ Internet Mail Server™ 4.0 (SIMS 4.0) is an enterprise-wide, open standards based, scalable electronic message handling system. The *Sun Internet Mail Server 4.0 Reference Manual* provides reference information about the Sun Internet Mail Server 4.0 product.

Use this guide as a companion to the *Sun Internet Mail Server 4.0 Administrator's Guide*. The administrator's guide focuses on how to configure, maintain, monitor, and troubleshoot Sun Internet Mail Server 4.0 using the Administration Console. The *Sun Internet Mail Server 4.0 Reference Manual* provides information about command-line utilities and configuration files. This information enables you to configure, maintain, monitor, and troubleshoot Sun Internet Mail Server 4.0.

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, "Commands Reference," is a reference to the server-side utilities used to configure and administer the Sun Internet Mail Server 4.0 product. The commands are listed by component. This chapter describes what each command does.

Chapter 2, "IMTA Configuration," describes IMTA configuration files that you can edit and that are supported by Sun Internet Mail Server 4.0.

Chapter 3, "Sun Directory Services Directory Information Tree and Schema," describes the Sun Directory Services configuration files.

Chapter 4, "SIMS Configuration Files," describes the `ims.cnf` file, the `sims.cnf` file, the `imdmc.cnf` file, and the `imta.cnf` file.

Appendix A, "Supported Standards," lists the industry standards that are supported by Sun Internet Mail Server 4.0.

"Glossary," The glossary covers terms that are specific or unique to Sun Internet Mail Server 4.0 and some terms that might be helpful to your understanding of this product.

Related Information

The following books are related to Sun Internet Mail Server 4.0. Included in this documentation set are:

- *Sun Internet Mail Server 4.0 Concepts Guide* – Provides a conceptual understanding of the SIMS product. By understanding how SIMS works on a conceptual level, readers will more easily understand the administrative tasks described in the *SIMS System Administration Guide* and *SIMS Reference Manual*.
- *Sun Internet Mail Server 4.0 Provisioning Guide* – Describes how to provision the SIMS LDAP directory with users, distribution lists, administrators, and domains by creating and importing LDIF records.
- *Sun Internet Mail Server 4.0 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 4.0 Administrator's Guide* – Describes how to fine-tune the default configuration, and maintain, monitor, and troubleshoot your mail server using the Administration Console, a GUI.

- *Sun Internet Mail Server 4.0 Delegated Management Guide* – Describes the SIMS Delegated Management Console and the tasks associated with the console. In particular, it describes how a delegated administrator for a hosted domain performs tasks on users and distribution lists.
- Reference manual pages (man pages) – Describe command-line utilities and detailed information about the arguments and attributes relevant to each command.
- *Sun Web Access Administrator's Guide* – Describes the core system administration tasks for Sun Web Access software.
- Sun Internet Mail Server 4.0 Release Notes – Covers open issues and late-breaking installation, administration, and reference information that is not published in the product books.
- Sun Internet Mail Server 4.0 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, product information, technical white papers, press coverage, and customer success stories.

What Typographic Changes Mean

Table P-1 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.	To delete a file, type: `rm` *filename*.
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

Table P-2 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 be performed only 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.

Notice

To better illustrate the process being discussed, SIMS manuals contain examples of data that might be used in daily business operations. The examples might include names of individuals, companies, brands, and products. SIMS manuals use only fictitious names, and any similarity to the names of individuals, companies, brands, and products used by any business enterprise is purely coincidental.

CHAPTER 1

Commands Reference

The following topics are covered in this chapter:

The command-line utilities described in this chapter allow you to configure and manage server resources for SIMS. Most of the utilities are located in `/opt/SUNWmail/sbin`. For complete information, see the corresponding man page for the command you want.

Note – To view the man page of a compound command, that is a command that consists of two more words such as "`imta test -rewrite`" or "`imadmin create user`," type `man word-word-word`. Example: man `imta-test-rewrite`

SIMS Administration Commands

The SIMS Administration command line (CLI) utilities are also called commands. (The SIMS Administrator uses all the commands. The Delegated Administrator uses the add, modify, delete, and search commands for the user and group objects.) Each command's task and object summary is provided in the imadmin man page. For specific information about the use of these commands, refer to the individual man pages.

TABLE 1-1 SIMS Administration Commands

Command	Description
`imadmin`	The imadmin(1M) man page describes the repertoire of SIMS Administration command line (CLI) utilities, also called commands.
`imadmin add admin`	Grants the SIMS Administrator privileges to a user. To grant privileges to multiple users, use the `-i` option.
`imadmin add group`	Adds a single group to the SIMS system. To add multiple groups, use the `-i` option. When a message is sent to the group address, SIMS sends the message to all members in the group.
`imadmin add ldapserver`	Adds a single ldap host:port for the admin server. Having multiple ldap servers for the admin server means that the admin server can failover to the next ldap server in the list when one ldap server goes down. To add multiple ldapservers with the same command, use the `-i` option.
`imadmin add user`	Adds a single user to the SIMS system. To add multiple users, use the `-i` option. You can run the `imadmin add user` command remotely using the DMS and supplying the domain name option.
`imadmin create domain`	Creates a single domain in the SIMS system. To create multiple domains, use the `-i` option.
`imadmin delete domain`	Deletes a single hosted domain from the SIMS system. To delete multiple hosted domains, use the -i option. When you invoke the command, the `simsDomainStatus` attribute of the domain's DC node entry is set to deleted. There is no undelete utility.
`imadmin delete group`	Deletes a single group from the SIMS system. To delete multiple groups, use the -i option. When you invoke this command, the `inetmailGroupStatus` of the group is set to deleted. There is no undelete utility.

TABLE 1-1 SIMS Administration Commands *(Continued)*

Command	Description
`imadmin delete user`	Deletes a single user from the SIMS system. To delete multiple users, use the `-i` option. When the hostname option is used by a Delegated Administrator, hostname refers to the Delegated Management Server (DMS) host. When used by a SIMS Administrator, hostname refers to the LDAP host. When you invoke this command, the `inetSubscriberStatus` of the user is set to deleted. The deleted user remains connected to the system until the purge task is run against the user. There is no undelete utility.
`imadmin modify currentldap`	Modifies the current ldap server property of the admin server. After this command is executed successfully, the admin GUI and CLI will use the specified ldap server for it's directory operations.
`imadmin modify domain`	Modifies attributes of a single domain's directory entry. To modify multiple domains, use the `-i` option.
`imadmin modify group`	Changes the attributes of a single group that already exists in the SIMS system. To change multiple groups, use the `-i` option.
`imadmin modify msglimits`	Changes the message limits attributes of a single existing channel. To change multiple channels, use the `-i` option.
`imadmin modify notary`	Changes the delivery status notification schedule of a single channel. To change multiple channels, use the `-i` option. For a permanent failure, the message is bounced and a notification is sent to the postmaster. For a transient failure, by default, the channel sends a maximum of three warning messages to the originator of the message.
`imadmin modify postmaster`	Returned messages: SIMS might be unable to deliver a message because of long-term service failures or invalid addresses. The IMTA channel program returns the message to the sender with an accompanying explanation of why the message was not delivered. Warning messages: The IMTA occasionally 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. The warning messages contain a description of what is wrong and how long delivery attempts will continue. To modify messages for multiple channels, use the `-i` option.
`imadmin modify user`	Changes the attributes of a single user that already exists in the SIMS system. To change multiple users, use the `-i` option. You can run the `imadmin modify user` command remotely using the Delegated Management Server and supplying the domain name option.
`imadmin purge domain`	Permanently deletes a single domain from the SIMS system. To permanently delete multiple domains, use the `-i` option. Use the command to remove all domains that have been deleted by the status attribute for a time period that is longer than the specified grace period. You can perform a purge at any time by invoking the command manually.

TABLE 1-1 SIMS Administration Commands *(Continued)*

Command	Description
`imadmin purge group`	Use the command to permanently delete all groups that have been deleted by the status attribute for a time period that is longer than the specified grace period. To permanently delete multiple groups, use the `-i` option. You can perform a purge at any time by invoking the command manually. There is no undelete utility.
`imadmin purge user`	Use the command to remove all users who have been deleted by the status attribute for a time period that is longer than the specified grace period. To permanently delete multiple users, use the `-i` option. You can perform a purge at any time by invoking the command manually.
`imadmin remove admin`	Removes SIMS Administrator privileges from a user. To remove SIMS Administrator privileges from multiple users, use the `-i` option.
`imadmin search admin`	SIMS Administrators use this command to search and display users who have SIMS administrative privileges. The `-n` domain name option will distinguish a Delegated Administrator when the option is specified. When the `-n` option is omitted, the command will filter for all users who have been granted SIMS Administrator privileges.
`imadmin search group`	Obtains all the LDAP attributes associated with a single group. To obtain all the LDAP attributes for multiple groups, use the `-i` option.
`imadmin search msglimits`	Searches for the message limits attributes of single existing channel. To search for attributes for multiple channels, use the `-i` option.
`imadmin search notary`	Searches for the delivery status notification schedule of a single channel. To perform a search of the schedules for multiple channels, use the `-i` option.
`imadmin search postmaster`	Returns the values of the variables, which direct the system how to treat failure and warning messages for a single postmaster. To return values for multiple postmasters, use the `-i` option.
`imadmin search user`	Obtains the LDAP attributes associated with one or more users.

The following commands are part of the SIMS Administration.

TABLE 1-2 SIMS Administration Commands - miscellaneous

Command	Description
`imedit`	Used when the specified configuration file (`config_file`) is locked according to the SIMS configuration file locking convention. The contents of `config_file` are copied to a temporary file in the same directory. The editor specified by the VISUAL or EDITOR environment variables is invoked on the temporary file. If the editor exits with status 0, the temporary file is renamed to the specified file name (`config_file`) and unlocked. If the editor exits with a non-zero status, the temporary file is removed and the specified file is unlocked.
`imxclean`	Checks for the existence of SIMS configuration file update logs that are not locked by any running process. If any are found, they are assumed to represent SIMS configuration file transactions that were interrupted before completion. `imxclean` examines the log and rolls the transaction forward or backs the transaction out, according to whether or not the transaction has been committed.
`setup-tty`	A script that installs the Sun Internet Mail Server (SIMS) and related files and packages onto the system.
`uninstall`	A script that removes the Sun Internet Mail Server (SIMS) and related files and packages from the system. You can specify uninstall to perform a standard or dramatic uninstall procedure. The dramatic uninstall option is a clean uninstall, removing all files installed by the SIMS installation process and created by SIMS during operation, except packages that may have already been present before the uninstall procedure

SIMS Monitoring

These commands let you monitor the components of SIMS. For the specifics concerning the use of these related commands, refer to the man pages.

TABLE 1-3 SIMS Monitoring Commands

Command	Description
`immonitor`	The umbrella script for monitoring the components of SIMS. For information about the installation, configuration, and example usage of these utilities look at the `Monitoring_ex_conf_scen.html`, `Monitoring_install.html`, and `Monitoring_intro.html`. These are found in `/opt/SUNWmail/html/C`.
`immonitor access`	Monitors the SIMS services, comprising Mail Delivery (SMTP), Message Access and Store (POP, IMAP), and Directory Service (LDAP). The Directory Service is monitored by looking up a specified user in the directory and measuring the response time. The Mail Delivery is monitored by sending a mail (SMTP) and the Message Access and Store is monitored by retrieving it. (POP/IMAP). This utility measures the response times of the various services and the total round trip time taken to send and retrieve a message.
`immonitor queue`	Monitors the IMTA component. Can be used to report the domains to which delivery has failed, the number of messages in the queue that are not eligible for delivery processing (messages with `.HELD` extension) and the number of messages in each channel queue.
`immonitor reenqueue`	Re-submits messages. The utility re-enqueues the message, for the movement to another channel to take place. The new channel with the rewrite rules must be created by the administrator prior to executing this command. Look at SIMS Administrator's Guide to create channel and add rewrite rules.
`immonitor system`	Monitors the status of system resources utilized by SIMS. These include: • the swap space • disk utilization • the number of connections in the ESTABLISHED state.
`immonitor users`	Reports user information including top <n> submitters and targets of a SPAM attack. Looking at this information the administrator can block connections from a spammers domain or move their messages to a slower channel.

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.

imbackup

Use the `imbackup` utility to back up stored Sun Message Store messages. `imbackup` should be run as the message store owner as specified in the ims.cnf file. The default owner should be set to inetmail.

Caution – The `imbackup` utility is unable to backup `/var/mail` messages, MIME files, and messages stored in the queues.

---

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.

`imdeluser`

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

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

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

Note – In SIMS 3.5 you needed to enter the full LDAP DN of the administrator. In Sun Internet Mail Server 4.0, you need to enter the login name (not the user name) of the administrator who has authority to manage the users in their domains.

`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.

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.

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.

iminitquota

The iminitquota utility reinitializes the user's in the user's mailbox based on their LDAP 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.

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.

---

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 can be changed.

imrestore

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

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.

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 file system is initialized.

Note – You cannot run this command after you have initialized a message store in `/var/opt/SUNWmail/ims`.

---

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 1-1 displays the structure.

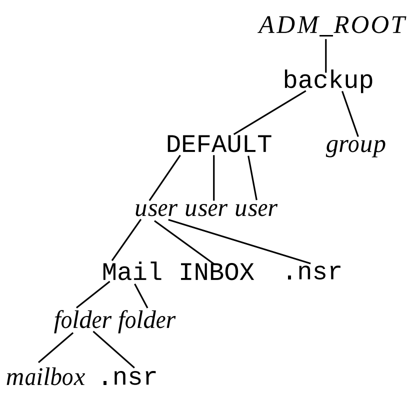


FIGURE 1-1 Backup directory hierarchy

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.

Sun Directory Services

This section summarizes the Sun Directory Services (LDAP) utilities commonly used for SIMS administration. For usage information on these utilities, refer to the Sun Directory Services documentation and the *SIMS Administrator's Guide*. For complete syntax and option information refer to the specific utility man pages.

Note – You can specify a regular expression for the distinguished name of an entry. For example, the regular expression `dn="cn=Joe Smith, ou=.*,dc=XYZ, dc=com,o=internet"` 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=.*"`.

---

dsserv

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

The `dsserv` daemon can be configured to provide replicated service for a data store, in conjunction with `slurpd`, the directory server update replication daemon. See the section "`dspushd`" on page 40 for details.

dsserv.conf

The file `dsserv.conf` contains configuration information for the dsservd daemon. This configuration information is also used by the `dspushd` and `dspulld` replication daemons and by the LDBM indexing utilities `ldif2ldbm`, `ldif2index`, `ldif2id2entry`, and `ldif2id2children`.

The `dsserv.conf` file consists of a series of global configuration options that apply to `dsservd` as a whole (including all data stores), followed by zero or more definitions that contain information specific to a data store.

dsserv.acl.conf

The file `dsserv.acl.conf` contains access control rules (also called ACLs) that apply to information stored in the directory. ACLs protect sensitive information such as user pass words. You can create extra ACLs that are specific to the kind of information that you need to protect. ACLs can be defined by using the Admin Console, or by hand, by editing the `dsserv.acl.conf` file. The syntax for an access control rule is:

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).

dsserv.at.conf

The file `dsserv.at.conf` contains the SIMS attributes.

dsserv.oc.conf

The file `dsserv.oc.conf` contains the SIMS object class.

dsserv.replog

The file `dsserv.replog` is produced by the stand-alone LDAP daemon, `dsservd`, 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 `dspushd`, the stand-alone LDAP update replication daemon. The records are separated by a blank line.

dsservcmd

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

dsprepush

The `dsprepush` 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, `dsprepush` generates replica entries for all databases and for all replica (slave) servers.

dspushd

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

Upon startup, `dspushd` reads the replication log (given either by the `replogfile` directive in the `dsserv` configuration file, or by the `-r` option). If the replication log file does not exist or is empty, `dspushd` 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 `dsserv` instances, `dspushd` locks the replication log, makes a private copy, releases the lock, and forks one copy of itself for each replica `dsserv` to be updated. Each child process binds to the slave `dsserv` with the DN given by the *binddn* option to the replica directive in the `dsserv config` file, and sends the changes. See `dsprepush` for details on the directory server daemon.

Note – By default, `dspushd` is set up not to run. To start it, you must modify `/etc/opt/SUNWconn/ldap/current/dsserv.ini` and change `startDspush=true`. Restart `dsserv`.

---

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 `dsserv.replog`. Entries created from the content of the LDIF file can be added to an LDAP directory using `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.

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.

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`.”

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.

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.

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.

ldbmcat

The `ldbmcat` command is used to convert a `dsserv` 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.

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.

`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 `dsserv`. 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`."

Internet Message Transfer Agent

The IMTA contains a modest collection of management utility programs that are used to perform various maintenance, testing, and management tasks. The following sections describe these utilities.

This section summarizes 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).

TABLE 1-4 IMTA Utilities

Utility	Description
`imta cache -close`	Has detached processes close their connections to the queue cache database.
`imta cache -rebuild`	Builds a new, synchronized queue cache database.
`imta cache -synch`	Synchronizes the current queue cache database.
`imta cache -view`	Views entries in the queue cache database.
`imta chbuild`	Compiles the IMTA character set conversion tables.
`imta clbuild`	Compiles an IMTA command definition file.
`imta cnbuild`	Compiles the IMTA configuration, alias, mapping, security, system wide filter, and option files
`imta counters -clear`	Clears the in-memory cache of channel counters
`imta counters -show`	Displays the contents of the database of channel counters
`imta counters -today`	Displays count of the number of messages processed today
`imta crdb`	Creates a IMTA database
`imta dirsync`	Recreates or updates the IMTA directory cache.
`imta find`	Finds the file name corresponding to the specified version of a IMTA file.
`imta process`	Lists currently running IMTA jobs.
`imta program`	Uses to manipulate the IMTA program delivery options.
`imta purge`	Purges IMTA log files.
`imta qm`	Manages IMTA message queues.
`imta queue`	Performs maintenance tasks on imta queue. The `imta queue retry_delivery channel_name` command reprocesses HELD messages in the channel specified by the `channel_name` parameter. To avoid mail loops, the IMTA holds messages when they have been forwarded more than 30 times. When corrected, the administrator can run this command to reprocess all the HELD messages. The `imta queue recover_crash` command rebuilds the MTA queue-cache database after a crash.
`imta renamedb`	Renames an IMTA database.
`imta restart`	Restarts detached IMTA processes.
`imta return`	Returns (bounce) a mail message to its originator.

TABLE 1-4 IMTA Utilities *(Continued)*

Utility	Description
imta run	Processes messages in a specified channel.
imta start	Starts detached IMTA processes.
imta stop	Shuts down the IMTA job controller and the IMTA Service Dispatcher.
imta submit	Processes messages in a specified channel.
imta submit_master	Process messages in a specified channel; on UNIX, a synonym for submit.
imta test -mapping	Test a mapping table.
imta test -match	Test a mapping wildcard pattern.
imta test -rewrite	Tests address rewriting.
imta view	Displays the contents of the specified "version" of a IMTA log file.
imta version	Prints the SIMS version number.

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 -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.

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.

---

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.

imta cache -view

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

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.

imta clbuild

The `imta clbuild` utility compiles a command line definition file and loads the resulting image file into shared memory. The IMTA ships with a pre-compiled command line definition image so it is not normally necessary to run this utility.

You must have superuser privileges to run this utility.

TABLE 1-5 clbuild cld-file-spec

Command Qualifiers	Defaults
`-debug`	`-nodebug`
`-image_file=file-spec`	`-noimage_file`
`-maximum`	`-nomaximum`
`-option_file=file-spec`	`-nooption_file`
`-remove`	None
`-sizes`	`-nosizes`
`-statistics`	`-nostatistics`

The file specification of a an IMTA command definition file to read as input; for example, `/opt/SUNWmail/imta/lib/imta.cld`.

Example

The standard command used to compile the basic IMTA command definition file is:

```
# imta clbuild -option_file -image_file=IMTA_COMMAND_DATA
/opt/SUNWmail/imta/lib/pmdf.cld
```

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 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

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.

`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`.

`imta counters -show`

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

`imta counters -today`

`imta counters -today` counts and displays the number of messages processed on this day. Note that the messages counted are the number of messages processed at the time that this command is executed.

Example

```
# imta counters -today
4263 messages processed so far today
30000 messages per day are permitted by your license
```

This example shows IMTA's count of the number of messages processed so far on a particular day.

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.

imta dirsync

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

The `-t` option executes `dirsync` in the test mode. It searches the directory and prints out the details on invalid entries, if there are any. No changes are made to the cache itself. Run this in conjunction with the `-F` option (causes the directory cache to be completely regenerated, thus creating a faithful image of the directory) to test the entire directory contents used by this MTA. Without the `-F` option, only the new additions are tested.

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

---

imta find

The `imta find` command finds the precise file name of the specified version of an IMTA file. IMTA log files contain a *-uniqueid,* which is appended to the file name to allow for the creation of multiple versions of the log file.

imta find understands these unique ids and can find the particular file name corresponding to the requested version of the file.

TABLE 1-6 imta - find file pattern

Command Qualifiers	Defaults
-f=offset-from-first	None
-l=offset-from-last	None

By default, if no offset qualifier (*n*) is specified, imta find locates the most recent version of the file.

Examples

```
# imta find /var/opt/SUNWmail/imta/log/tcp_local_slave.log
```

This command will print out the file name of the /imta/log/tcp_local_slave.log-uniqueid file most recently created.

```
# imta find /imta/log/tcp_smtp_server.log -f=0
```

This command will display the file name of the oldest /var/opt/SUNWmail/imta/log/tcp_local_slave.log-uniqueid file.

imta process

This command displays the current IMTA processes. 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.

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.

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.

imta qm: Queue Management

The imta qm command is a utility for inspecting and manipulating the channel queue directories and the messages contained in them. It has some functionality overlap with the imta cache, imta queue, and imta counters commands. Privileges sufficient to read, create, and delete files in the channel queue directory tree as well as read and update the queue cache database are required to use this.

For example, imta queue -retry_delivery can be achieved using the release command in imta qm. As another example, some of the information returned by imta cache -view is also available through the directory command in imta qm. However, imta qm does not completely replace imta cache or imta queue.

imta qm can only be run by root or inetmail.

To run imta qm in interactive mode, issue the command

```
$ imta qm
```

To run imta qm in non-interactive mode, issue a command such as:

```
$ imta qm <command>
```

Use the exit or quit command to exit imta qm. The commands accepted by this utility in maintenance mode are summarized in .

Note – Some of the commands available in the interactive mode are not available in the non-interactive mode and the reverse is also true.

TABLE 1-7 imta qm Mode Commands

Commands	Descriptions
counters	Controls aspects of the channel counter caches and database.
date	Shows current date and time
delete	Irrevocably deletes the specified messages
directory	Lists currently queued messages

TABLE 1-7 `imta qm` Mode Commands *(Continued)*

Commands	Descriptions
exit	Exits the utility.
held	Lists messages which have been marked as held.
help	Obtains help.
history	Displays message delivery history information.
hold	Marks a message as held.
quit	Exits the utility.
read	Displays message envelope and header information
release	Releases held message.
return	Returns a message to its originator.
run	Executes commands from the specified file.
view	Controls whether the channel queue directory tree or queue cache database is viewed.

`imta queue`

Use the `imta queue` command 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).

`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.

▼ To create or update the IMTA databases:

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. 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.

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.

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

---

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 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.

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.

imta start

The `imta start` command starts up detached IMTA processes. If no component parameter is specified, then the IMTA Job Controller and 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 using 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.

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.

imta submit

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

imta test -mapping

Use the `imta test -mapping` utility to test the behavior of a mapping table in the mapping file. The result of mapping an input string will be output along with information about any meta characters specified in the output string.

If an input string is supplied on the command line, then only the result of mapping that input string will be output. If no input string is specified, `imta test -mapping` will enter a loop, prompting for an input string, mapping that string, and prompting again for another input string. `imta test -mapping` will exit when you press `CTRL-D`.

TABLE 1-8 `imta test -mapping` syntax

Command Qualifiers	Defaults
`-flags=list of characters`	`-noflags`
`-image_file`	`-image_file`
`-mapping_file`=file-spec	`-mapping_file=IMTA_MAPPING_FILE`
`-option_file`=file-spec	`-option_file=IMTA_OPTION_FILE`
`-table`=table-name	None

Example

In the following example, the sample PAGER mapping is tested. The `-mapping_file` qualifier is used to select the mapping file `pager_table.sample` instead of the default mapping file.

```
% pmdf test -mapping -noimage_file
-mapping_file=/imta/table/pager_table.sample
Enter table name: PAGER
Input string: H|From: "Dancer" <dan@bridge.com> (Doof City)
Output string: H|F:dan
Output flags: [0,1,2,89]
Input string: ^D
%
```

imta test -match

You can use `imta test -match` to test a mapping pattern, particularly, to test wildcard and glob matching.

When invoked, `imta test -match` prompts for a pattern and then for a target string to compare against the pattern, and will output whether or not the target string matched and if it did match, which characters in the target string matched which wildcard or glob of the pattern. `imta test -match` will loop, prompting for input, until you exit using a `CTRL/D`.

Example

In the following example, the sample mapping pattern `$[ax1]*@*.bridge.com` is tested for several sample target strings.

```
% imta test -match
Pattern: $[ax1]*@*.bridge.com
Target: xx11a@sys1.bridge.com
Match.
0 - xx11a
1 - sys1
Pattern: $[ax1]*@*.bridge.com
Target: 12a@node.bridge.com
No match.
Pattern: $[ax1]*@*.bridge.com
Target: 1xa@node.bridge.com
Match.
0 - 1xa
1 - node
Pattern: ^D
%
```

imta test -rewrite

Use `imta test -rewrite` to provide 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 you press `CTRL-D`.

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.

imta view

Use `imta view` to display a specified version of an IMTA log file. IMTA log files contain a `uniqueid`, which is appended to the file name to allow creation of multiple versions of the log file. `imta view` understands these unique ids and can display the contents of the particular file corresponding to the requested version of the file.

By default, if no offset qualifier (n) is specified, `imta view` displays the most recent version of the file.

TABLE 1-9 `imta view` Command Qualifiers

Command Qualifier	Description
`-f=offset-from-first`	Use this qualifier to specify displaying the nth version of the file (starting counting from 0). For instance, to display the earliest (oldest) version of the file, specify `-f=0`.
`-l=offset-from-last`	Use this qualifier to specify displaying the nth from the last version of the file (starting decrementing from 0 as the most recent version). For instance, to display the most recent (newest) version of the file, specify -l=0.

`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.

Installation

This section describes the utilities associated with the installation process. For more information on installation, refer to the *Sun Internet Mail Server 4.0 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_4_0/products/sims/setup-tty`.

The `setup-tty` 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 `setup-tty` appear in TABLE 1-10.

TABLE 1-10 `setup-tty` options

Option	Description
`-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.
`-c removeall`	Specifies a dramatic `removeall` of SIMS and related files and packages. This option removes data and configuration files left over from the standard `removeall`. This is a clean `removeall`, removing all files installed by the SIMS installation process and created by SIMS during operation, with the exception of packages that were present before the `removeall` procedure.
`-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 1-11 for a description of the parameters included in the `sims_setup.dat` file.

Examples

The following command performs a standard interactive installation, with `-c install` as the default parameter:

```
% 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

TABLE 1-11 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 1-11 sims_setup.dat File

Parameter	Description
administrator-name	The user name for the directory administrator.
administrator-passwd	The password for the directory administrator.
cgi-bin	The location of the CGI bin directory for the HotJava Views server.
dcRoot	Root node for the directory tree (default is internet).
do_upgrade	Specifies whether or not to upgrade to SIMS 3.5 (1=upgrade, 0=do not upgrade).
document-root	The location of the document root for the Sun Web Server.
fax-number	The fax number. This is a text entry.
filename	Name of the file that contains the SIMS license.
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.
ha_masterlhost	The logical host name for the HA installation.
ha_sharedfs	The shared disk location for the HA installation.
hostname	Fully qualified name of the local host.
install-mode	Specifies which SIMS product package to upgrade (3=optional features install, 4=core install).
install-sws	Determines whether or not to install the Sun web server (0=do not install, 1=install).
ldap-port	The LDAP port number.

TABLE 1-11 `sims_setup.dat` File *(Continued)*

Parameter	Description
`ldap-server`	Hostname of the LDAP master server.
`ldap-type`	ldap server selected by user: `sun` for SunDS and `netscape` for Netscape DS
`ldap_up`	1 if the ldap (directory) is local; or if it is remote and the remote machine has the directory server running. 0 if the remote ldap server is not running.
`locality`	The name of the locality. This is a text entry.
`maildomain`	The mail domain.
`mta-role`	Determines if the IMTA is installed behind the firewall (0=not behind firewall, 1=behind firewall).
`org-name-long`	The name of the organization. This is a text entry.
`phone-number`	The phone number. This is a text entry.
`postal-address`	The postal address. This is a text entry.
`postmaster`	The postmaster's ID.
`postmaster_uid`	The postmaster's UID.
`province`	The name of the province. This is a text entry.
`readfromfile`	Determines whether or not the directory server license will be read from a local file (0=no, 1=yes).
`remote-ldap`	Determines where the LDAP server is located (0=local host, 1=not local).
`remadmin`	Determines whether or not to install the remote administrator option (0=do not install, 1=install).
`rootdomain`	The root domain.
`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).
`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).
`sims-doc`	1 if the SIMS doc must be installed. 0 in all other cases.
`siteadmin-name`	The user name for the SIMS administrator.

TABLE 1-11 sims_setup.dat File *(Continued)*

Parameter	Description
siteadmin-passwd	The password for the SIMS administrator.
spmServer	The fully qualified host name of the machine where SPM is installed (default: local host).
smarthost	Has a text string value only if mta-role=1 (behind the firewall).
varmail	Determines whether or not the /var/mail message store is supported (0=not supported, 1=supported).
upgrade-possible	0=not upgradable.
webaccess	Determines whether or not to install the WebAccess option (0=do not install, 1=install).
ws_port	The Web server's port (default is 80).

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 might not remove packages that have been installed by a separate application and might 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. To start sendmail, the user must either reboot the system or manually start the sendmail program.

Web server packages can be removed by uninstall, but httpd is not stopped.

Syntax

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

The options for this command appear in TABLE 1-12.

TABLE 1-12 `uninstall` options

`-c sims`	Specifies a standard `uninstall` of SIMS and related files and packages. The standard `uninstall` does not remove the directories and files of configuration and data, for example, 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.

Note – Option `-d` is recommended before re-installing SIMS.

Examples

The following command performs a standard `uninstall`:

```
% uninstall -c sims
```

The following command performs a dramatic `uninstall`:

```
% uninstall -d sims
```

CHAPTER **2**

IMTA Configuration

The following topics are covered in this chapter:

The IMTA Configuration Files

This section explains the structure and layout of the IMTA configuration files. Some configuration modifications can be done using the command-line interface, as described in Chapter 1 in "Internet Message Transfer Agent" on page 43, or by using the accompanying SIMS Admin Console, as described in the *SIMS Administrator's Guide*. Modifications not possible through either can be done by editing the configuration files. We recommend that only experienced administrators edit and modify the configuration files. Many configuration settings can be defined using the GUI described in the *Sun Internet Mail Server System Administrator's Guide*.

Caution – Sun does not guarantee that the changes made by modifying configuration files will be recorded properly by the administration console. The administration console interfaces cannot recognize different types of information that the user might add to the configuration files.

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 most files is limited to 252 characters and you can split a logical line into multiple physical lines using the backslash (\) continuation character.

Note – If you change any of the files manually, 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` configuration file is synchronized with the administration console.

By preprocessing these files and storing them in memory, the initialization time for IMTA is significantly reduced, thereby improving IMTA's performance.

TABLE 2-1 lists the IMTA configuration files with a short description.

TABLE 2-1 IMTA Configuration files

File	Description	Page
Autoreply Option File	Options used by the `autoreply` program.`/etc/opt/SUNWmail/imta/autoreply.opt`	187
Alias File (mandatory)	Implements aliases not present in the directory. `/etc/opt/SUNWmail/imta/aliases`	136
Channel Options File	Many channels use channel options files to set channel specific options. `/etc/opt/SUNWmail/imta/`*`channel`*`_option`	139
Conversion File	Used by conversion channel to control message body part conversions. `/etc/opt/SUNWmail/imta/conversions`	147
Dirsync Option File (mandatory)	Options used by the `dirsync` program. `/etc/opt/SUNWmail/imta/dirsync.opt`	186
Dispatcher Configuration File (mandatory)	Configuration file for dispatcher. (Enterprise Edition only). `/etc/opt/SUNWmail/imta/dispatcher.cnf`	195
IMTA Configuration File (mandatory)	Used for address rewriting and routing as well as channel definition. `/etc/opt/SUNWmail/imta/imta.cnf`	68
Mapping File (mandatory)	Repository of mapping tables. (Enterprise Edition only). `/etc/opt/SUNWmail/imta/mappings`	156
IMTA Option File	File of global IMTA options. `/etc/opt/SUNWmail/imta/option.dat`	171
IMTA Tailor File (mandatory)	File to specify locations and some tuning parameters. `/etc/opt/SUNWmail/imta/imta_tailor`	182
Job Controller Config. File (mandatory)	Configuration file used by the `job_controller`. `/etc/opt/SUNWmail/imta/job_controller.cnf`	188
Log Files (mandatory)	`mail.log` file to indicate the message traffic through the IMTA and log files for specific master or slave programs. `/var/opt/SUNWmail/imta/log/*`	155
Message Files (mandatory)	Enqueued messages are stored in message files in channel queue directories. `/var/opt/SUNWmail/imta/queue/*/*`	105

TABLE 2-2 lists the IMTA database files with a short description.

TABLE 2-2 IMTA Database Files

File	Description
Address Reversal Database (mandatory)	Used to change addresses in outgoing mail. This database is created using the `imta dirsync` command and is not editable directly. DO NOT EDIT. `/var/opt/SUNWmail/imta/db/reversedb*`
Alias Database (mandatory)	Implements aliases, mail forwarding, and mailing lists. Changes should be made to the directory and running `imta dirsync`. DO NOT EDIT. `/var/opt/SUNWmail/imta/db/aliasesdb*`
Domain Database	Used for Storing additional rewriting rules. DO NOT EDIT. `/var/opt/SUNWmail/imta/db/domaindb`
General Database	Used with domain rewriting rules or in mapping rules, for site-specific purposes. Also used for POP before SMTP support. `/var/opt/SUNWmail/imta/db/generaldb`
Profile Database (mandatory)	Database to store program delivery, file delivery, and other special delivery mechanism information. This database is also created from information in the directory during `imta dirsync`. DO NOT EDIT. `/var/opt/SUNWmail/imta/db/profiledb*`
Queue Cache Database (mandatory)	The messages currently enqueued are recorded in the queue cache database. Channel master programs determine which messages to process by querying this database. DO NOT EDIT. `/var/opt/SUNWmail/imta/queue_cache/*`

`imta.cnf` 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. An individual channel definition forms 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.

Domain Rewriting Rules

Domain rewriting rules, or, as they are also called, “rewrite rules,” play two important roles.

- They are used to rewrite addresses into their proper form.
- 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 configuring rules, refer to the *Sun Internet Mail Server 4.0 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 “Configuration File Format” on page 195). 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 *usertemplate*, 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]
```

TABLE 2-3 describes the parts of the rewriting rule structure.

TABLE 2-3 Rewriting Rule Structure

Part	Description
`pattern`	The rule applies if the pattern matches the domain part of the address. Patterns can contain wildcards.
`controls`	The applicability of a rule can be limited using these control sequences. Control sequences can be located either before the user template or after the routing system. The selection criteria are described in TABLE 2-8. They include: • Envelope or header addresses • Direction (To or From) • Source or destination channel of the message
[*userTemplate*]	Specifies how the user part of the address is rewritten. The template can 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 2-5.
%	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* can 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 “Template Substitutions” on page 77 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 can 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 nonroutable.

Extracting the First Host or Domain Specification

The process of rewriting an address starts by extracting the first host or domain specification from the address. (Readers not familiar with RFC 822 address conventions are advised to read that standard 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 is 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 examples of addresses and the host names that could be extracted first are shown in TABLE 2-4.

TABLE 2-4 Extracted Addresses and Host Names

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	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 user names; 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 or 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 (that is, the left 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 or domain specification does not match any pattern, in which case it is said to "not match any rule", the first part of the host or 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 or 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. All probes that contain asterisks are done only in the configuration file rewrite rules table; the domain database is not checked. This process proceeds until either a match is found or the entire host or 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 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, nonasterisk 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. 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 nonzero length, then you return to step 1. If the resulting string has zero length (for example, was previously “.”) then the lookup process has failed and you 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
*.*.*.*
.
```

Rewrite rule templates

Once the host/domain specification matches a rewrite rule, it is rewritten using the template part of the rule. The template specifies three things:

1. A new user name for the address,
2. a new host/domain specification for the address, and
3. the name of a system attached to an existing IMTA channel (the “routing system”) to which messages to this address should actually be sent.

The usual format for templates is A%B@C, where A is the new user name, B is the new host/domain specification, and C is the routing system. If B and C are identical, %B can be omitted; for example, you may use A@C when B and C are identical.

Substitution Strings in Templates

Substitution strings are allowed in the template. Any occurrences of `$U` in the template are replaced with the user name from the original address, any occurrences of `$H` are replaced with the portion of the host/domain specification that was not matched by the rule, and any occurrences of `$D` are replaced by the portion of the host/domain specification that was matched by the rewrite rule. $L substitutes the portion of a domain literal that was not matched by the rewrite rule.

Note – User names of the form a."b" will be replaced by "a.b" because current Internet standardization work is deprecating the former syntax from RFC 822. It is expected that the latter usage will become mandatory in future.

`$$` expands to a single dollar sign, `$`; *$%* expands to a single percent, `%` (the percent is not interpreted as a template field separator in this case); and `$@` expands to a single at sign, `@` (also not interpreted as a field separator).

As an example, suppose that the host/domain specification `jdoe@stream.com` has matched the rewrite rule

```
stream.com          $U@STREAM.COM
```

Then the template will produce the user name jdoe, the host/domain specification `STREAM.COM`, and the routing system `STREAM.COM`. In a slightly more complicated example, assume that the host/domain specification has matched the rewrite rule

```
.com               $U%$H$D@TCP-DAEMON
```

In this case, `$U` = `jdoe`, `$H` = `stream`, and `$D` = `.com`. The template produces the username jdoe, the host/domain specification stream.com, and the routing system TCP-DAEMON.

TABLE 2-5 on page 78 contains a summary of these and other substitution strings, which are presented in “Template Substitutions” on page 77.

Finishing the Rewriting Process

One of two things can happen once the host/domain specification is rewritten.

- If the routing system is not associated with the local channel or there are no additional host/domain specifications in the address, the rewritten specification is substituted into the address replacing the original specification that was extracted for rewriting, and the rewriting process terminates.
- If the routing system matches the local channel and there are additional host/domain specifications that appear in the address, the rewritten address is discarded, the original (initial) host/domain specification is removed from the

address, a new host/domain specification is extracted from the address and the entire process is repeated. Rewriting will continue until either all the host/domain specifications are gone or a route through a non-local channel is found. This iterative mechanism is IMTA's way of providing support for source routing. In effect, superfluous routes through the "local system" are removed from addresses by this process.

Rewrite Rule Failure

If a host/domain specification fails to match any rewrite rule and no default rule is present, IMTA uses the specification "as-is"; for example, the original specification becomes both the new specification and the routing system. If the address has a nonsensical host/domain specification it will be detected when the routing system does not match any system name associated with any channel and the message will be bounced.

Syntax checks after rewriting

No additional syntax checking is done after the rewrite rules have been applied to an address. This is deliberate—it makes it possible for rewrite rules to be used to convert addresses into formats that do not conform to RFC 822. However, this also means that mistakes in the configuration file may result in messages leaving the IMTA with incorrect or illegal addresses.

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@stream.com
```

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

A summary of template substitutions appears in TABLE 2-5.

TABLE 2-5 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`	User name from original address.
`$$`	Inserts a literal dollar sign ($).
`$%`	Inserts a literal percent sign (%).
`$@`	Inserts a literal at sign (@).
`$\`	Forces material to lowercase.
`$^`	Forces material to uppercase.
`$_`	Uses original case.
`$W`	Substitutes in a random, unique string.
`$[...]`	Invokes customer-supplied routine; substitutes in result.
`$(text)`	General database substitution; rule fails if lookup fails.
`${...}`	Applies specified mapping to supplied string.
`$&n`	The *nth* part of unmatched (or wildcard) host, counting from left to right, starting from 0.
`$!n`	The *nth* part of unmatched (wildcard) host, as counted from right to left, starting from 0.
`$*n`	The *nth* part of matching pattern, counting from left to right, starting from 0.
`$#n`	The *nth* part of matching pattern, 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 */
```

The `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 the IMTA 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 2-8 causing this rule to be used only for forward-pointing addresses):

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

A forward-pointing address, `jdoe@stream.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. The keyword `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.

The `$M` *channel* causes the rule to fail if the channel *channel* is not currently doing the rewriting. The `$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 possibly can act only in conjunction with the channel to which the message is being queued. This is useful if 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. The keyword `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.

The $Q *channel* causes the rule to fail if the channel *channel* is not the destination. The $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 refers 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 is 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 might require 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 2-6.

TABLE 2-6 Single Field Substitutions

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

```
*.STREAM.COM     $U%$&0.stream.com@mailhub.stream.com
```

Then the result from the template will be `jdoe@vaxa.stream.com` with `mailhub.stream.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. 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 or 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 or 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 rescanned 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@stream.decnet` matches the following rewrite rule:

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

Then, the text string `stream` will be looked up in the general database and the result of the lookup, if any, used for the rewrite rule's template. Suppose that the result of looking up `stream` is `$u%eng.stream.com@tcp-local`. Then the output of the template will be `jdoe@eng.stream.com` (`username` = `jdoe`, host or domain specification = `eng.stream.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 files specified with the `IMTA_GENERAL_DATABASE` option in the `/etc/opt/SUNWmail/imta/imta/tailor` file, which are usually the files `/var/opt/SUNWmail/imta/db/generaldb.*`.

Applying 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 user name 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.

Special Patterns and Tags

Rewrite rules can make use of several special patterns, summarized in TABLE 2-7, and discussed in the following subsections.

TABLE 2-7 Patterns for Rewrite Rules

Pattern	Description/Usage
$%	Percent Hack Rule. Matches any host / domain specification of the form A%B.
$!	Bang-style Rule. Matches any host / domain specification of the form A!B.
.	Match-all Rule. Matches any host / domain specification.
$[}	Matches any domain literal address. For example, *joe@[129.165.12.11}*
$*	Matches any address. This is applied before any other rule.

In addition to these special patterns, IMTA also has the concept of *tags*, which may appear in rewrite rule patterns. These tags are used in situations where an address may be rewritten several times and, based upon previous rewritings, distinctions must be made in subsequent rewrites by controlling which rewrite rules match the address.

A Rule to Match Percent Hacks

If IMTA tries to rewrite an address of the form `A%B` and fails, it tries one extra rule before falling through and treating this address form as `A%B@localhost`. This extra rule is the *percent hack rule*. The pattern is `$%`. The pattern never changes. This rule is only activated when a local part containing a percent sign has failed to rewrite any other way (including the match all rule described below).

The percent hack rule is useful for assigning some special, internal meaning to percent hack addresses.

A Rule to Match Bang-Style (UUCP) Addresses

If IMTA tries to rewrite an address of the form `B!A` and fails, it tries one extra rule before falling through and treating this address form as `B!A@localhost`. This extra rule is the *bang-style rule*. The pattern is `$!`. The pattern never changes. This rule is only activated when a local part containing an exclamation point has failed to rewrite any other way (including the default rule described below).

The bang-style rule can be used to force UUCP style addresses to be routed to a system with comprehensive knowledge of UUCP systems and routing.

A Rule to Match Any Address

The special pattern "." (a single period) will match any host/domain specification if no other rule matches and the host/domain specification cannot be found anywhere in the channel table. In other words, the *"." rule* is used as a last resort when address rewriting would fail otherwise.

Note – When the match-all rule matches and its template is expanded, `$H` expands to the full host name and `$D` expands to a single dot "`.`". Thus, `$D` is of limited use in a match-all rule template!

Tagged Rewrite Rule Sets

As the rewrite process proceeds it may be appropriate to bring different sets of rules into play. This is accomplished by the use of the rewrite rule tag. The current tag is prepended to each pattern before looking it up in the configuration file or domain database. The tag can be changed by any rewrite rule that matches by using the `$T` substitution string in the rewrite rule template (described below).

Tags are somewhat sticky; once set they will continue to apply to all hosts that are extracted from a single address. This means that care must be taken to provide alternate rules that begin with the proper tag values once any tags are used. In practice this is rarely a problem since tags are usually used in only very specialized applications. Once the rewriting of the address is finished the tag is reset to the default tag---an empty string.

By convention all tag values end in a vertical bar |. This character is not used in normal addresses and thus is free to delineate tags from the rest of the pattern.

Changing the Current Tag Value, $T

The `$T` control sequence is used to change the current rewrite rule tag. The rewrite rule tag is prepended to all rewrite rule patterns before they are looked up in the configuration file and domain database. Text following the `$T`, up until either an @ sign, percent sign, `$N`, `$M`, `$Q`, `$C`, `$T`, or `$?` is taken to be the new tag.

Tags are useful in handling special addressing forms where the entire nature of an address is changed when a certain component is encountered. For example, suppose that the special host name `internet`, when found in a source route, should be removed from the address and the resulting address forcibly matched against the TCP-DAEMON channel. This could be implemented with rules like the following (localhost is assumed to be the official name of the local host):

```
internet        $S$U@localhost$Tmtcpforce |
mtcp-force|.    $U%$H@TCP-DAEMON
```

The first rule will match the special host name `internet` if it appears in the source route. It forcibly matches `internet` against the local channel, which insures that it will be removed from the address. A rewrite tag is then set. Rewriting proceeds, but no regular rule will match because of the tag. Finally, the default rule is tried with the tag, and the second rule of this set fires, forcibly matching the address against the TCP- DAEMON channel regardless of any other criteria.

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 or 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:

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

only applies to (rewrite) `envelope To:` addresses of the form `user@stream.com`. If a domain or 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:

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

which results in messages to `user@stream.com` being passed to the directory channel. However, should the verify channel rewrite a message with the address `user@stream.com`, that message does not again pass through the verify channel. This then allows all mail to `user@stream.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 appears in TABLE 2-8.

TABLE 2-8 Template Control Sequences

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

Handling Large Numbers of Rewrite Rules

IMTA always reads in all the rewrite rules from the configuration file and stores them in memory in a hash table. Use of a compiled configuration bypasses the overhead associated with reading the configuration file each and every time the information is needed; a hash table is still used to store all of the rewrite rules in memory. This scheme is adequate for small to medium numbers of rewrite rules. However, some sites may require as many as 10,000 rewrite rules or more, which can consume prohibitive amounts of memory.

The IMTA solves this problem by providing an optional facility for storing large numbers of rewrite rules in an ancillary indexed data file. Whenever the regular configuration file is read, IMTA checks for the existence of the domain database, IMTA_DOMAIN_DATABASE. If this database exists, it is opened and consulted whenever an attempted match fails on the rules found in the configuration file. The domain database is only checked if a given rule is not found in the configuration file, so rules can always be added to the configuration file to override those in the database. By default, the domain database is used to store rewrite rules associated with hosted domains. DO NOT EDIT BY HAND.

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 example, issue the following command:

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

For a detailed description of the `imta test -rewrite` utility, refer to **Chapter 1, "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 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. No blank lines can appear among the lines of rewrite rules. Lines with comments (beginning with an exclamation point in the first column) are 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.

TABLE 2-9 lists the routing and queuing of messages by the preceding configuration:

TABLE 2-9 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

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 or 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 2-10 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 "Channel Configuration Keywords," and to TABLE 2-10.

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.

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 2-10; the keywords shown in **`boldface`** are defaults.

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 whether you have any keywords in your configuration file that don't match a known rights list identifier.

TABLE 2-10 Channel Keywords

Keyword	Usage
`addrsperfile` (page 106)	Number of addresses per message file.
`addrsperjob` (page 105)	Number of addresses to be processed by a single job.
`after`	Specifies time delay before master channel programs run.
`allowetrn`	The IMTA will attempt to honor all ETRN commands. (default)
`allowswitchchannel` (page 118)	Allows switching to this channel from an **`allowswitchchannel`** channel.
`bangoverpercent` (page 100)	Group `A!B%C` as `A!(B%C).`
`bidirectional` (page 102)	Channel is served by both a master and slave program.
`blocketrn`	Tells the IMTA not to honor ETRN commands.
`blocklimit` (page 131)	Maximum number of IMTA blocks allowed per message.
`cacheeverything` (page 103)	Caches all connection information.
`cachefailures` (page 155)	Caches only connection failure information.
`cachesuccess` (page 103)	Caches only connection success information.
`charset7` (page 121)	Default character set to associate with 7-bit text messages.
`charset8` (page 121)	Default character set to associate with 8-bit text messages.
`checkehlo` (page 113)	Checks the SMTP response banner for whether to use `EHLO.`
`commentinc` (page 127)	Leaves comments in message header lines intact.
`commentomit` (page 127)	Removes comments from message header lines.
`commentstrip` (page 127)	Removes problematic characters from comment field in message header lines.
`commenttotal` (page 127)	Strips comments (material in parentheses) everywhere.
`connectalias` (page 101)	Does not rewrite addresses upon message dequeue.

TABLE 2-10 Channel Keywords *(Continued)*

Keyword	Usage
connectcanonical (page 101)	Rewrites addresses upon message dequeue.
copysendpost (page 109)	Sends copies of failures to the postmaster unless the originator address is blank.
copywarnpost (page 110)	Sends copies of warnings to the postmaster unless the originator address is blank.
daemon	Specifies the name of a gateway to which the daemon is routed.
datefour (page 128)	Converts date/time specifications to four-digit years.
datetwo (page 128)	Converts date/time specifications to two-digit years.
dayofweek (page 128)	Includes day of week in date and time specifications.
defaultmx (page 115)	Channel determines whether or not to do MX lookups from network.
deferred (page 108)	Honors deferred delivery dates.
defragment (page 130)	Reassembles any MIME-compliant message/partial parts queued to this channel.
domainetrn	Tells the IMTA to honor only those ETRN commands that specify a domain. It also causes the IMTA not to echo back the name of the channel that the domain matched and that the IMTA will be attempting to run.
domainvrfy (page 114)	Issues SMTP VRFY commands using full address.
ehlo (page 113)	Uses EHLO on all initial SMTP connections.
eightbit (page 120)	Channel supports 8-bit characters.
eightnegotiate (page 120)	Channel should negotiate use of eight bit transmission, if possible.
eightstrict (page 120)	Channel should reject messages that contain unnegotiated 8-bit data.
errsendpost (page 109)	Sends copies of failures to the postmaster if the originator address is illegal.
errwarnpost (page 110)	Sends copies of warnings to the postmaster if the originator address is illegal.
expandchannel	
expandlimit (page 107)	Processes an incoming message “offline” when the number of addressees exceeds this limit.
exproute (page 100)	Explicit routing for this channel’s addresses.
filesperjob (page 105)	Number of queue entries to be processed by a single job.
forwardcheckdelete	Tells the IMTA to do a forward lookup after each reverse lookup and to ignore (delete) the reverse lookup returned name if the forward lookup of that name does not match the original connection IP address. Use the original IP address instead.
forwardchecknone	No forward lookup is done
forwardchecktag	Tells the IMTA to do a forward lookup after each reverse lookup and to tag the IP name with an asterisk, *, if the number found using the forward lookup does not match that of the original connection.

TABLE 2-10 Channel Keywords *(Continued)*

Keyword	Usage
`headerinc`	Places the message header at the top of the message.
`headerlabelalign` (page 129)	Aligns headers.
`headerlinelength` (page 129)	Folds long headers.
`headerread` (page 123)	Applies header trimming rules from an options file to the message headers upon message enqueue (use with caution).
`headertrim` (page 123)	Applies header trimming rules from an options file to the message headers (use with caution).
`identnone` (page 116)	Does not perform IDENT lookups; does perform IP-to-hostname translation.
`identnonelimited`	Has the same effect as identnone as far as IDENT lookups, reverse DNS lookups, and information displayed in Received: header. With identnonelimited the IP literal address is always used as the basis for any channel switching due to use of the switchchannel keyword, regardless of whether the DNS reverse lookup succeeds in determining a host name.
`identnonenumeric` (page 116)	Does not perform IDENT lookups or IP-to-hostname translation.
`identnonesymbolic`	Disables this `IDENT` lookup, but does do IP to host name translation; only the host name will be included in the `Received: header` for the message.
`identtcp` (page 116)	Performs IDENT lookups on incoming SMTP connections and IP to host name translation.
`identtcplimited`	Has the same effect as identtcp as far as IDENT lookups, reverse DNS lookups, and information displayed in Received: header. With identtcplimited the IP literal address is always used as the basis for any channel switching due to use of the switchchannel keyword, regardless of whether the DNS reverse lookup succeeds in determining a host name.
`identtcpnumeric` (page 116)	Performs IDENT lookups on incoming SMTP connections, but do not perform IP to hostname translation.
`identtcpsymbolic`	Tells the IMTA to perform a connection and lookup using the `IDENT` protocol (RFC 1413).
`ignoreencoding` (page 124)	Ignores Encoding: header on incoming messages.
`immediate` (page 102)	Delivery started immediately after submission for messages of second-class or higher priority.
`immnonurgent` (page 102)	Delivery started immediately after submission, even for messages with lower-than-normal priority.
`immnormal` (page 102)	Delivery started immediately after submission for messages of normal-or-higher priority.
`immurgent` (page 102)	Delivery started immediately after submission for urgent messages only.
`improute` (page 100)	Implicit routing for this channel’s addresses.
`includefinal`	Include final form of address in delivery notifications.
`inner` (page 122)	Rewrites inner message headers.

TABLE 2-10 Channel Keywords *(Continued)*

Keyword	Usage
innertrim (page 123)	Applies header trimming rules from an options file to inner message headers (use with caution).
interpretencoding (page 124)	Interprets Encoding: header on incoming messages.
lastresort (page 116)	Specifies a last-resort host.
linelength (page 121)	Message lines exceeding this length limit are wrapped.
linelimit (page 131)	Maximum number of lines allowed per message.
localvrfy (page 114)	Issues SMTP VRFY command using local address.
logging (page 132)	Log message enqueues and dequeues into the log file.
mailfromdnsverify	Setting on an incoming TCP/IP channel causes the IMTA to verify that an entry in the DNS exists for the domain used on the SMTP MAIL FROM: command, and to reject the message if no such entry exists.
master (page 102)	Channel is served only by a master program.
master_debug (page 132)	Generates debugging output in the channel's master program output.
maxblocks (page 130)	Maximum number of IMTA blocks per message; longer messages are broken into multiple messages.
maxheaderaddrs (page 129)	Maximum number of addresses per message header line; longer header lines are broken into multiple header lines.
maxheaderchars (page 129)	Maximum number of characters per message header line; longer header lines are broken into multiple header lines.
maxjobs (page 105)	Maximum number of jobs that can be created at one time.
maxlines (page 130)	Maximum number of message lines per message; longer messages are broken into multiple messages.
maxperiodic--	The maxperiodicnonurgent, maxperiodicnormal, or maxperiodicurgent keywords specify the maximum priority of message that a periodic job should try to deliver; the job will ignore messages of higher priority.
maxprocchars (page 132)	Specifies maximum length of headers to process.
maysaslserver	Causes the SMTP server to permit clients to attempt to use SASL authentication.
minperiodic--	The minperiodicnonurgent, minperiodicnormal, or minperiodicurgent keywords specify the minimum priority of message that a periodic job should try to deliver; the job will ignore messages of lower priority.
missingrecipientpolicy	Takes an integer value specifying the approach to use for such messages; the default value, if the keyword is not explicitly present, is 0, meaning that envelope To: addresses are placed in a To: header.
multiple (page 106)	Accepts multiple destination hosts in a single message copy.
mustsaslserver	Causes the SMTP server to insist that clients use SASL authentication; the SMTP server will not accept messages unless the remote client successfully authenticates.
mx (page 115)	TCP/IP network and software supports MX record lookups.

TABLE 2-10 Channel Keywords *(Continued)*

Keyword	Usage
nobangoverpercent (page 100)	Group A!B%C as (A!B)%C (default).
nocache (page 155)	Does not cache any connection information.
nodayofweek (page 128)	Removes day of week from date/time specifications.
nodeferred (page 108)	Does not honor deferred delivery dates.
nodefragment (page 130)	Does not perform special processing for message/partial messages.
noehlo (page 113)	Never use the SMTP EHLO command.
noexproute (page 100)	No explicit routing for this channel's addresses.
noheaderread (page 123)	Does not apply header trimming rules from option file upon message enqueue.
noheadertrim (page 123)	Does not apply header trimming rules from options file.
noimproute (page 100)	No implicit routing for this channel's addresses.
noinner (page 122)	Does not rewrite inner message headers.
noinnertrim (page 123)	Does not apply header trimming to inner message headers.
nologging (page 132)	Does not log message enqueues and dequeues into the log file.
nomailfromdnsverify	Means that IMTA does not verify that an entry in the DNS exists for the domain used.
nomaster_debug (page 132)	Does not generate debugging output in the channel's master program output.
nomx (page 115)	TCP/IP network does not support MX lookups.
nonrandommx (page 115)	Does MX lookups; does not randomize returned entries with equal precedence.
nonurgentblocklimit (page 103)	Forces messages above this size to wait unconditionally for a periodic job.
noreceivedfor (page 125)	Does not include envelope to address in Received header.
noreceivedfrom	Instructs the IMTA to construct Received: headers without including the original envelope From: address.
noremotehost (page 119)	Uses local host's domain name as the default domain name to complete addresses.
norestricted	Does not apply RFC 1137 restricted encoding to addresses.
noreverse (page 122)	Does not apply reverse database to addresses.
normalblocklimit (page 103)	Forces messages above this size to nonurgent priority.
nosasl	SASL authentication will not be permitted or attempted.
nosaslserver	SASL authentication will not be permitted.
nosendpost (page 109)	Does not send copies of failures to the postmaster.
nosendetrn	The IMTA will not send an ETRN command.
noserviceall (page 133)	Indicates that the master program should only process the messages that were queued to process after its inception.
noslave_debug (page 132)	Does not generate debugging output in the channel's slave program output.
nosmtp (page 112)	Channel does not use SMTP.

TABLE 2-10 Channel Keywords *(Continued)*

Keyword	Usage
noswitchchannel (page 118)	Stays with the server channel; do not switch to the channel associated with the originating host; does not permit being switched to.
notices (page 108)	Specifies the amount of time that may elapse before notices are sent and messages returned.
novrfy (page 114)	Does not issue SMTP VRFY commands.
nowarnpost (page 110)	Does not send copies of warnings to the postmaster.
nox_env_to (page 124)	Does not add X-Envelope-to header lines while enqueuing.
period (page 102)	Specifies periodicity of periodic channel service.
periodic (page 102)	Channel is serviced only periodically; immediate delivery processing is never done.
personalinc (page 127)	Leaves personal names in message header lines intact.
personalomit (page 127)	Removes personal name fields from message header lines.
personalstrip (page 127)	Strips problematic characters from personal name fields in message header lines.
port (page 115)	Sends to the specified TCP/IP port.
postheadbody (page 111)	Both the message's header and body are sent to the postmaster when a delivery failure occurs.
postheadonly (page 111)	Only the message's header is sent to the postmaster when a delivery failure occurs.
queue (page 108)	Specifies queue master channel programs run in.
randommx (page 115)	Does MX lookups; randomizes returned entries with equal precedence.
receivedfor (page 125)	Includes envelope to address in Received header.
receivedfrom	Instructs the IMTA to include the original envelope From: address when constructing a Received: header for an incoming message if the IMTA has changed the envelope From: address due to, for example, certain sorts of mailing list expansions. receivedfrom is the default.
remotehost (page 119)	Uses remote host's name as the default domain name to complete addresses.
restricted (page 122)	Applies RFC 1137 restricted encoding to addresses.
returnenvelope (page 125)	Controls use of blank envelope return addresses.
reverse (page 122)	Applies reverse database to addresses.
saslswitchchannel	Causes incoming connections to be switched to a specified channel upon a client's successful use of SASL. It takes a required value, specifying the channel to which to switch.
sendpost (page 109)	Sends copies of failures to the postmaster.
sendetrn	Tells the IMTA to send an ETRN command, if the remote SMTP server says it supports ETRN. The sendetrn keyword should be followed by the name of the system requesting that its messages receive a delivery attempt.

TABLE 2-10 Channel Keywords *(Continued)*

Keyword	Usage
sensitivity--	The sensitivitynormal, sensitivitypersonal, sensitivityprivate, and **sensitivitycompanyconfidential** (messages of any sensitivity are allowed) keywords set an upper limit on the sensitivity of messages that can be accepted by a channel.
serviceall (page 133)	Specifies that the master program should attempt to process all messages queued to the channel each time it runs.
sevenbit (page 120)	Channel does not support 8-bit characters; 8-bit characters must be encoded.
silentetrn	Tells the IMTA to honor all ETRN commands, but without echoing the name of the channel that the domain matched and that the IMTA will be attempting to run.
single (page 106)	Only one envelope To address per message copy.
single_sys (page 106)	Each message copy must be for a single destination system.
slave (page 102)	Channel is serviced only by a slave program.
slave_debug (page 132)	Generates debugging output in the channel's slave program output.
smtp (page 112)	Channel uses SMTP.
smtp_cr (page 112)	Accepts CR as an SMTP line terminator.
smtp_crlf (page 112)	Requires CRLF as the SMTP line terminator.
smtp_lf (page 112)	Accepts LF as an SMTP line terminator.
sourceroute	Uses source routes in the message envelope; synonymous with 822.
subdirs (page 107)	Uses multiple subdirectories.
suppressfinal	Causes IMTA to suppress the final address form, if an original address form is present, from notification messages.
switchchannel (page 118)	Switches from the server channel to the channel associated with the originating host.
threaddepth (page 112)	Number of messages triggering new thread with multithreaded SMTP client.
unrestricted (page 122)	Does not apply RFC 1137 restricted encoding to addresses.
urgentblocklimit (page 103)	Forces messages above this size to normal priority.
usereplyto (page 126)	Specifies mapping of Reply-to header.
useresent (page 126)	Specifies mapping of Resent- headers for non-RFC 822 environments.
vrfyallow	Tells IMTA to issue a detailed, informative response.
vrfydefault	Tells IMTA to provide a detailed, informative response, unless the channel option HIDE_VERIFY=1 has been specified.
vrfyhide	Tells IMTA to issue only a vague, ambiguous response.
warnpost (page 110)	Sends copies of warnings to the postmaster.
x_env_to (page 124)	Adds X-Envelope-to header lines while enqueuing.

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 mailers can interpret addresses using the latter set of conventions, it does not say that such an interpretation is required. 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 you want 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. Another scenario occurs when IMTA connects to a system through a channel that cannot perform proper routing for itself. In this case, all addresses associated with other channels need to have routing indicated 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. The default, noimproute, 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.

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. The keyword 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 reflects 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.

The default behavior (immediate immnormal) enables immediate processing for all but nonurgent or lower priority messages.

Delivery by 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, although 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 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 time and delay other messages. The cache normally records both connection successes and failures. (Successful connection attempts are recorded 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, an 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. The `cachefailures` enables caching of connection failures but not successes. Finally, `cachesuccesses` caches only successful connections. This last keyword is equivalent to `nocache` for channels.

Priority of Messages Handled by Periodic Jobs (`minperiodicnonurgent`, `minperiodicnormal`, `minperiodicurgent`, `maxperiodicnonurgent`, `maxperiodicnormal`, `maxperiodicurgent`)

When periodic delivery jobs are used they normally process all messages queued for the channel. However, on some channels you might want to limit normal periodic job processing to only messages of specified priorities. Other special site-supplied periodic jobs may then process the remaining messages. For instance, a site might choose to have normal IMTA periodic jobs pass over nonurgent messages, leaving the nonurgent messages to be delivered by a site-supplied job (perhaps scheduled to run at off-peak hours).

The `minperiodicnonurgent`, `minperiodicnormal`, or `minperiodicurgent` keywords specify the minimum priority of message that a periodic job should try to deliver; the job will ignore messages of lower priority.

The `maxperiodicnonurgent`, `maxperiodicnormal`, or `maxperiodicurgent` keywords specify the maximum priority of message that a periodic job should try to deliver; the job will ignore messages of higher priority.

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 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. 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 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 establishes a connection only 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. The keyword `single` means that a separate copy of the message should be created for each destination address on the channel. The keyword `single_sys` creates a single copy of the message for each destination system used.The keyword `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.

The keyword 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. The keyword `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.

The keyword `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. The keyword `postheadbody` returns both the headers and the contents of the message. It is the default.The keyword `postheadonly` causes only the headers to be sent to the postmaster.

Including Altered Addresses in Notification Messages (`includefinal`, `suppressfinal`)

When IMTA generates a notification message (bounce message, delivery receipt message, and so on), there may be both an "original" form of a recipient address and an altered "final" form of that recipient address available to IMTA. IMTA always includes the original form (assuming it is present) in the notification message, because that is the form that the recipient of the notification message (the sender of the original message, which the notification message concerns) is most likely to recognize.

The `includefinal` and `suppressfinal` channel keywords control whether IMTA also includes the final form of the address. Suppressing the inclusion of the final form of address may be of interest to sites that are "hiding" their internal mailbox names from external view; such sites may prefer that only the original, "external" form of address be included in notification messages. `includefinal` is the default and includes the final form of the recipient address. `suppressfinal` causes IMTA to suppress the final address form, if an original address form is present, from notification messages.

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. The keyword 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 might 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. The nosmtp is the default.

The keyword smtp is mandatory for all SMTP channels. The keywords smtp_cr, smtp_crlf, and smtp_lf can be used on SMTP channels to specify the character sequences to accept as line terminators. The keyword smtp_crlf means that lines must be terminated with a carriage return (CR) line feed (LF) sequence. The keyword 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 using 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 noncompliant 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. The keyword noehlo disables all use of the EHLO command. The keyword 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.

Receiving an SMTP ETRN Command (allowetrn, blocketrn, domainetrn, silentetrn)

The allowetrn, blocketrn, domainetrn, and silentetrn keywords control the IMTA response when a sending SMTP client issues the SMTP ETRN command, requesting that the IMTA attempt to deliver messages in the IMTA queues. allowetrn is the default; the IMTA will attempt to honor all ETRN commands. silentetrn tells the IMTA to honor all ETRN commands, but without echoing the name of the channel that the domain matched and that the IMTA will be attempting to run. blocketrn tells the IMTA not to honor ETRN commands. domainetrn tells the IMTA to honor only ETRN commands that specify a domain; it also causes the IMTA not to echo back the name of the channel that the domain matched and that the IMTA will be attempting to run.

Sending an SMTP ETRN Command (sendetrn, nosendetrn)

The extended SMTP command ETRN (RFC 1985) allows an SMTP client to request that a remote SMTP server start up processing of the remote side's message queues destined for sending to the original SMTP client; that is, it allows an SMTP client and SMTP server to negotiate "switching roles", where the side originally the sender becomes the receiver, and the side originally the receiver becomes the sender. In other words, ETRN provides a way to implement "polling" of remote SMTP systems for messages incoming to one's own system. This can be useful for systems that have only transient connections between each other, for example, over dial-up lines. When the connection is brought up and one side sends to the other, using the ETRN command, the SMTP client can also tell the remote side that it should now try to deliver any messages that needs to travel in the reverse direction.

The SMTP client specifies on the SMTP ETRN command line the name of the system to which to send messages (generally the SMTP client system's own name). If the remote SMTP server supports the ETRN command, it will trigger execution of a separate process to connect back to the named system and send any messages awaiting delivery for that named system.

The sendetrn and nosendetrn channel keywords control whether the IMTA SMTP client sends an ETRN command at the beginning of an SMTP connection. The default is nosendetrn, meaning that the IMTA will not send an ETRN command. The sendetrn keyword tells the IMTA to send an ETRN command, if the remote SMTP server says it supports ETRN. The sendetrn keyword should be followed by the name of the system requesting that its messages receive a delivery attempt.

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.

Responding to SMTP VRFY commands (vrfyallow, vrfydefault, vrfyhide)

These keywords control the IMTA SMTP server's response when a sending SMTP client issues an SMTP VRFY command. The vrfyallow keyword tells IMTA to issue a detailed, informative response. The vrfydefault tells IMTA to provide a detailed, informative response, unless the channel option HIDE_VERIFY=1 has been specified. The vrfyhide keyword tells IMTA to issue only a vague, ambiguous response. These keywords allow per-channel control of VRFY responses, as opposed to the HIDE_VERIFY option, which normally applies to all incoming TCP/IP channels handled through the same SMTP server.

TCP/IP Port Number (port)

The SMTP over TCP/IP channels normally connects to port 25 when sending messages. The port keyword can be used to instruct an SMTP over TCP/IP channel to connect to a nonstandard 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. The keyword randommx specifies that MX lookups should be done and MX record values of equal precedence should be processed in random order. The keyword 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 might 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. The keyword defaultmx is the default on channels that support MX lookups in any form.

Specifying a Last Resort Host (lastresort)

The lastresort keyword is used to specify a host to connect even 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, identtcplimited, identtcpnumeric, identtcpsymbolic, identnone, identnonelimited, identnonenumeric, identnonesymbolic, forwardchecknone, forwardchecktag, forwardcheckdelete)

The identtcp keyword tells the 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: header of the message, with the host name corresponding to the incoming IP number, as reported from a DNS reverse lookup and the IP number itself.

The identtcpsymbolic keyword tells the 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: header of the message, with the actual incoming IP number, as reported from a DNS reverse lookup; the IP number itself is not included in the Received: header.

The identtcpnumeric keyword tells the 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: header 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 for the IDENT lookup caused by identtcp or identtcpnumeric to be useful.

Be aware that IDENT query attempts may incur a performance hit. Increasingly routers will “black hole” attempted connections to ports that they don’t recognize; if this happens on an IDENT query, then the IMTA does not hear back until the connection times out (a TCP/IP package controlled time-out, typically on the order of a minute or two).

A lesser performance factor occurs when comparing identtcp or identtcpsymbolic to identtcpnumeric. The DNS reverse lookup called for with identtcp or identtcpsymbolic incurs some additional overhead to obtain the more user-friendly host name.

The identnone keyword disables this IDENT lookup, but does do IP to host name translation, and both IP number and host name will be included in the Received: header for the message. The identnonesymbolic keyword disables this IDENT lookup, but does do IP to host name translation; only the host name will be included in the Received: header for the message. The identnonenumeric keyword disables this IDENT lookup and inhibits the usual DNS reverse lookup translation of IP number to host name, and might result in a performance improvement at the cost of less user-friendly information in the Received: header. identnone is the default.

The identtcplimited and identnonelimited keywords have the same effect as identtcp and identnone, respectively, as far as IDENT lookups, reverse DNS lookups, and information displayed in Received: header. Where they differ is that with identtcplimited or identnonelimited the IP literal address is always used as the basis for any channel switching due to use of the switchchannel keyword, regardless of whether the DNS reverse lookup succeeds in determining a host name.

The forwardchecknone, forwardchecktag, and forwardcheckdelete channel keywords can modify the effects of doing reverse lookups, controlling whether the IMTA does a forward lookup of an IP name found using a DNS reverse lookup, and if such forward lookups are requested what the IMTA does if the forward lookup of the IP name does not match the original IP number of the connection. The forwardchecknone keyword is the default, and means that no forward lookup is done. The forwardchecktag keyword tells the IMTA to do a forward lookup after each reverse lookup and to tag the IP name with an asterisk, *, if the number found using the forward lookup does not match that of the original connection. The forwardcheckdelete keyword tells the IMTA to do a forward lookup after each reverse lookup and to ignore (delete) the reverse lookup returned name if the forward lookup of that name does not match the original connection IP address. Use the original IP address instead.

Note – Having the forward lookup not match the original IP address is normal at many sites, where a more “generic” IP name is used for several different IP addresses.

These keywords are only useful on SMTP channels that run over TCP/IP.

Selecting 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). The keyword `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 host name. Consequently, this keyword is useful for setting up anti-spamming, but it may affect performance.

Host Name 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 @stream.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. The keyword noremotehost is the default.

The switchchannel keyword as described, in the preceding section, Selecting 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 noncompliant clients are in use) rather than attempting to fix the network-wide problem on your IMTA hosts.

Legalizing Messages Without Recipient Headers (missingrecipientpolicy)

RFC 822 (Internet) messages are required to contain a recipient header: a To:, Cc:, or Bcc: header. A message without such a header is illegal. Nevertheless, some broken user agents and mailers (for example, many older versions of sendmail) will allow illegal messages.

The missingrecipientpolicy keyword takes an integer value specifying the approach to use for such messages; the default value, if the keyword is not explicitly present, is 0, meaning that envelope To: addresses are placed in a To: header.

TABLE 2-11 `missingrecipientpolicy` Values

Value	Action
0	Place envelope To: recipients in a To: header.
1	Pass the illegal message through unchanged.
2	Place envelope To: recipients in a To: header.
3	Place all envelope To: recipients in a single Bcc: header.
4	Generate a group construct (for example, ;) To: header, To: Recipients not specified.
5	Generate a blank Bcc: header.
6	Reject the message.

Note that the MISSING_RECIPIENT_POLICY option can be used to set an IMTA system default for this behavior.

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 be done only on character sets specified in the character set definition file `charsets.txt` found in the IMTA table directory, `/imta/table/charsets.txt`. Use the names defined in this file, 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. 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 interpreted only 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. The keyword `noinner` tells IMTA not to rewrite inner message header lines. It is the default. The keyword `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. The keyword `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. The keyword `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` using 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`, `receivedfrom`, `noreceivedfrom`)

The `receivedfor` keyword instructs the IMTA that if a message is addressed to just one envelope recipient, to include that envelope to the address in the `Received:` header it constructs. The keyword `receivedfor` is the default. The `noreceivedfor` keyword instructs the IMTA to construct `Received` headers without including any envelope addressee information.

The `receivedfrom` keyword instructs the IMTA to include the original `envelope From`: address when constructing a `Received`: header for an incoming message if the IMTA has changed the `envelope From`: address due to, for example, certain sorts of mailing list expansions. `receivedfrom` is the default. The `noreceivedfrom` keyword instructs the IMTA to construct `Received`: headers without including the original `envelope From`: address.

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 2-12 indicates the mapping specifications for the `Reply-to:` header.

TABLE 2-12 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 Using a Gateway to Non-RFC 822 Environments (`useresent`)

The `useresent` keyword controls the use of `Resent-` headers when using a gateway to environments that do not support RFC 822 headers. This keyword takes a single integer-valued argument. TABLE 2-13 lists the values used for mapping the `Resent-` headers.

TABLE 2-13 `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 to rewrite and eliminate short form addresses and otherwise convert them to legal addresses. During this process, comments (strings enclosed in parentheses) are extracted and may 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. The `commentinc` keyword tells IMTA to retain comments in header lines. It is the default. The keyword `commentomit` tells IMTA to remove any comments from addressing headers, for example, `To`, `From`, or `Cc` headers.

The keyword `commenttotal` tells IMTA to remove any comments from all headers, including `Received:` headers; this keyword is not normally useful or recommended. `commentstrip` tells IMTA to strip any nonatomic 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 short form 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. The keyword `personalinc` tells IMTA to retain personal names in the headers. It is the default. The keyword `personalomit` tells IMTA to remove all personal names.The keyword `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. The keyword `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 – The keyword `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. The keyword `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 – The keyword `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@stream.com
From:     mary@stream.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)

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. The keyword `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. The keyword `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. The keyword `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.

The keyword 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.

The keyword 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, 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 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, nologging)

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.

Delivery of Deferred Messages (serviceall, noserviceall)

Master programs normally process only a subset of the messages queued for the channel. There may be other messages that were queued to the channel at some prior time that will not be processed. However, on some channels, particularly those that only provide a link to a single mail component, this sort of operation may be inappropriate: if the immediate delivery job is successful in connecting to the mail component it may be able to easily process all the messages that are queued.

The serviceall and noserviceall keywords control this behavior. noserviceall, the default, indicates that the master program should only process the messages that were queued to process after its inception. serviceall specifies that the master program should attempt to process all messages queued to the channel each time it runs.

It may be tempting to indulge in use of serviceall on most or all channels. Be warned, however, that use of serviceall is probably not suitable for most channels that connect to multiple remote systems, or channels that entail lots of per-message overhead. If serviceall is used on such channels it may cause a dramatic increase in network and message processing overhead and the net result may be slower message processing overall.

Note that these keywords do not change the order in which message processing occurs. Immediate jobs always attempt to process the messages they were created to process prior to turning to other messages that are also in the channel queue.

Sensitivity checking (sensitivitynormal, sensitivitypersonal, sensitivityprivate, sensitivitycompanyconfidential)

The sensitivity checking keywords set an upper limit on the sensitivity of messages that can be accepted by a channel. The default is sensitivitycompanyconfidential; messages of any sensitivity are allowed through. A message with no Sensitivity: header is considered to be of normal, that is, the lowest, sensitivity. Messages with a higher sensitivity than that specified by such a keyword will be rejected when enqueued to the channel with an error message:

```
message too sensitive for one or more paths used
```

Note that IMTA does this sort of sensitivity checking at a per-message, not per-recipient, level: if a destination channel for one recipient fails the sensitivity check, then the message bounces for all recipients, not just for those recipients associated with the sensitive channel.

SMTP AUTH (maysaslserver, mustsaslserver, nosasl, nosaslserver, saslswitchchannel)

The maysaslserver, mustsaslserver, nosasl, nosaslserver, and saslswitchchannel channel keywords are used to configure SASL (SMTP AUTH) use during the SMTP protocol by SMTP channels such as TCP/IP channels.

nosasl is the default and means that SASL authentication will not be permitted or attempted. It subsumes nosaslserver, which means that SASL authentication will not be permitted. Specifying maysaslserver causes the SMTP server to permit clients to attempt to use SASL authentication. Specifying mustsaslserver causes the SMTP server to insist that clients use SASL authentication; the SMTP server will not accept messages unless the remote client successfully authenticates.

Use saslswitchchannel to cause incoming connections to be switched to a specified channel upon a client's successful use of SASL. It takes a required value, specifying the channel to which to switch.

Verify the Domain on MAIL FROM: Is In the DNS (mailfromdnsverify, nomailfromdnsverify)

Setting mailfromdnsverify on an incoming TCP/IP channel causes the IMTA to verify that an entry in the DNS exists for the domain used on the SMTP MAIL FROM: command, and to reject the message if no such entry exists. nomailfromdnsverify is the default and means that no such check is performed.

Note that performing DNS checks on the return address domain may result in rejecting some valid messages (for example, from legitimate sites that have not yet registered their domain name, or at times of bad information in the DNS); it is contrary to the spirit of being generous in what you accept and getting the e-mail through, expressed in RFC 1123, Requirements for Internet Hosts. However, some sites might want to perform such checks in cases where junk email (SPAM) is being sent with forged email addresses from non-existent domains.

Domain Database

The IMTA dirsync program creates in the domain database as well as in a file, /etc/opt/SUNWmail/imta/domains.rules. You can use this file instead of the domain database in case the number of domains is significantly fewer.

Note – Incremental dirsync does not update the domains.rules file. If you use the file instead of the database, the newly added domains would be recognized only after the next full dirsync. Also, if you make any changes to the domains.rules file manually, those changes will be overwritten by imta dirsync.

Aliases

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 providing 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@stream.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 apply only to addresses mapped to the local channel. Since the only addresses truly considered to match a channel are `Envelope To` addresses, aliases can apply only 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.

The Alias Database

The IMTA uses the information in the directory and creates the alias database. The alias database is consulted once each time the regular alias files is consulted. However, the alias database is checked before the regular alias file is used. In effect, the database acts as a sort of address rewriter that is invoked prior to using the alias file. Refer to the *SIMS Provisioning Guide* for information on what directory attributes are used to create user and distribution list entries in the alias database.

Note – The format of the database itself is private. Do not try to edit the database directly. Make all required changes in the directory.

Alias File

The alias 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 users in hosted domains)

user: <address> (for users in non-hosted domains. For example,
default-domain)
```

For example:

```
! A /var/mail/ user
inetmail : inetmail@native-daemon

! A message store user
ms_testuser : mstestuser@sims-ms-daemon
```

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.

By default, the file `/etc/opt/SUNWmail/imta/aliases/usr` is included. This file is updated by the `imta dirsync` program. List of addresses too long to fit in the alias database are put in this file.

Local Channel

The local channel (l) is unique because addresses diverted to it are looked up in the alias table. In general, the result of the alias table lookup matches another channel, causing the message to be enqueued to this channel. In practice, no message is enqueued to the local channel.

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.

Native Channel

The native channel is used to deliver messages to /var/mail mailboxes.

var/mail 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 `native_option` (for example, `/etc/opt/SUNWmail/imta/native_option`).

Option files consist of several lines. Each line contains the setting for one option. An option setting has the form:

```
option=value
```

The *value* may be either a string or an integer, depending on the option's requirements.

TABLE 2-14 Local Channel Options

Options	Descriptions
FORCE_CONTENT_LENGTH (0 or 1; UNIX only)	If FORCE_CONTENT_LENGTH=1, then the IMTA adds a Content-length: header line to messages delivered to the native channel, and causes the channel not to use the ">From" syntax when "From" is at the beginning of the line. This makes local UNIX mail compatible with Sun's newer mail tools, but potentially incompatible with other UNIX mail tools.
REPEAT_COUNT (integer) SLEEP_TIME (integer)	In case the user's new mail file is locked by another process when the IMTA tries to deliver the new mail, these options provide a way to control the number and frequency of retries the local channel program should attempt. If the file can not be opened after the number of retries specified, the messages will remain in the local queue and the next run of the local channel will attempt to deliver the new messages again. The REPEAT_COUNT option controls how many times the channel programs will attempt to open the mail file before giving up. REPEAT_COUNT defaults to 30, (30 attempts). The SLEEP_TIME option controls how many seconds the channel program waits between attempts. SLEEP_TIME defaults to 2 (two seconds between retries).

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
```

The `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 2-15.

TABLE 2-15 SMTP Channel Options

Option	Description
`ALLOW_ETRNS_PER_SESSION` (integer)	Sets a limit on the number of ETRN commands accepted per session. The default is 1.
`ALLOW_TRANSACTIONS_PER_SESSION` (Integer)	Sets a limit on the number of messages allowed per connection. The default is no limit.
`ALLOW_RECIPIENTS_PER_TRANSACTION` (Integer)	Sets a limit on the number of recipients allowed per message. The default is no limit.
`ATTEMPT_TRANSACTIONS_PER_SESSION` (Integer)	Sets a limit on the number of messages IMTA will attempt to transfer during any one connection session.
`COMMAND_RECEIVE_TIME` (Integer)	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)	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).
`DATA_RECEIVE_TIME` (Integer)	Specifies, in minutes, how long to wait to receive data during an SMTP dialogue. The default is 60.
`DATA_TRANSMIT_TIME` (Integer)	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 2-15 SMTP Channel Options *(Continued)*

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)	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. 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.
LOG_BANNER (0 or 1)	The LOG_BANNER option controls whether the remote SMTP server banner line is included in mail.log* file entries when the logging channel keyword is enabled for the channel. A value of 1 (the default) enables logging of the remote SMTP server banner line; a value of 0 disables it.

TABLE 2-15 SMTP Channel Options *(Continued)*

Option	Description
LOG_CONNECTION (integer)	The LOG_CONNECTION option controls whether or not connection information, e.g., the domain name of the SMTP client sending the message, is saved in mail.log file entries and the writing of connection records when the logging channel keyword is enabled for the channel. This value is a decimal integer representing a bit-encoded integer, the interpretation of which is given below: Bit-0 Value-1: When set, connection information is included in E and D log records. Bit-1 Value-2: When set, connection open/close/fail records are logged by message enqueue and dequeue agents such as the SMTP and X.400 clients and servers. Bit-2 Value-4: When set, I records are logged recording ETRN events. Where Bit 0 is the least significant bit. This channel option defaults to the setting of the global IMTA option LOG_CONNECTION as set in the IMTA option file. This channel option may be set explicitly to override on a per-channel basis the behavior requested by the global option.
LOG_TRANSPORTINFO (0 or 1)	The LOG_TRANSPORTINFO controls whether transport information, such as the sending and receiving side IP addresses and TCP ports, is included in mail.log file entries when the logging channel keyword is enabled for the channel. A value of 1 enables transport information logging. A value of 0 disables it. This channel option defaults to the setting of the global IMTA option LOG_CONNECTION as set in the IMTA option file.
MAIL_TRANSMIT_TIME (Integer)	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 set up. This option controls the number of threads per process. The default if this option is not specified is 10.
RCPT_TRANSMIT_TIME (Integer)	Specifies, in minutes, how long to spend transmitting the SMTP command RCPT TO. The default is 10.

TABLE 2-15 SMTP Channel Options *(Continued)*

Option	Description
STATUS_DATA_RECEIVE_TIME (Integer)	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)	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)	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)	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.
STATUS_MAIL_RECEIVE_TIME (Integer)	Specifies, in minutes, how long to wait to receive the SMTP response to a sent MAIL FROM command. (Also corresponds to the time we wait for the greetings.) The default is 10.
STATUS_RCPT_RECEIVE_TIME (Integer)	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)	Specifies, in minutes, how long to wait to receive the SMTP response to general SMTP commands, (commands other than those with specified time out values set using other specifically named options). The default value is 10.
STATUS_TRANSMIT_TIME (Integer)	Specifies, in minutes, how long to spend transmitting the SMTP response to an SMTP command.
TRACE_LEVEL (0, 1, or 2)	This option controls whether TCP/IP level trace is included in debug log files. The default value is 0, meaning that no TCP/IP packet traces are included; a value of 1 tells IMTA to include TCP/IP packet traces in any debug log files; a value of 2 tells IMTA to include DNS lookup information as well as TCP/IP packet traces.

The Pipe Channel

The pipe channel performs delivery of messages using per-user, site-supplied programs. It provides a similar functionality to `sendmail`'s pipe (`|`). The following differences are designed so that they will not pose a security threat. First, delivery programs to be invoked by the pipe channel must be registered by the system administrator. This registration is done performed using the `imta program` utility. See "imta program" on page 51 for information about `imta program`.

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 an 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 using the command corresponding to this name. The attribute `maildeliveryoption` must have one value equal to `program`.

For example, to add a UNIX command `myprocmail` as a program that can be invoked by the users, the user's LDAP entry should contain the following attributes/values:

```
maildeliveryoption: program
mailprogramdeliveryinfo: myprocmail
```

See alternative delivery programs in the *SIMS Administrator's Guide* for more information.

The Hold Channel

The hold channel is used to hold the messages of a recipient temporarily halted from receiving new messages. Messages may be halted because a user's name is being changed, or their mailbox is being moved from one mailhost or domain to another. There may also be other reasons to temporarily halt a user from receiving messages, but these are the most common.

Messages are placed in the hold channel in two ways:

1. Setting one of the `maildeliveryoption` values of a user to `hold`. All other `maildeliveryoption` values are ignored (`maildeliveryoption` is a multi-valued attribute), and messages to the user are routed to the hold channel.
2. Executing the `hold_slave` program. This program steps through all other channels and moves the existing messages whose recipient(s) matches those specified by the arguments into the hold channel. (See the `hold_slave` man page.)

Unlike most channels, the hold channel master program is not configured to run automatically. Messages queued in the hold channel will remain there until the `hold_master` program is invoked by the administrator. (See the `hold_master` man page.)

To migrate user, first mark the user as being moved (use `imadmin modify user` to set `maildeliveryoption` to `hold`). Then invoke `hold_slave` to move any messages already in the other queues to the hold queue. At this point, perform the remaining migration steps. Once you have completed these steps, remove `maildeliveryoption=hold`, and then invoke `hold_master` to reenqueue messages to their proper channels.

For more information, refer to the man pages for `hold_master`, `hold_slave`, and `imadmin-modify-user`.

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 using 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
```

The *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 that do not originate from the `tcp_intranet` channel and that are going 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 of the Conversion Channel

Configuration of the conversion channel in the IMTA configuration file (`imta.cnf`) is performed by default. An 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"
```

Conversion Control Parameters

The rule parameters currently provided are shown in TABLE 2-16. Parameters not listed in the table are ignored.

TABLE 2-16 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	Inputs A1-Format from enclosing MESSAGE/RFC822 part.
IN-A1-TYPE	Inputs A1-Type from enclosing MESSAGE/RFC822 part.
IN-CHAN	Inputs 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.

TABLE 2-16 Conversion Parameters *(Continued)*

Parameter	Description
IN-CHANNEL	Synonym for IN-CHAN.
IN-DESCRIPTION	Inputs MIME Content-Description.
IN-DISPOSITION	Inputs MIME Content-Disposition.
IN-DPARAMETER-DEFAULT-*n*	Inputs 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*	Inputs MIME Content-Disposition parameter name whose value is to be checked; *n* = 0, 1, 2, ...
IN-DPARAMETER-VALUE-*n*	Inputs MIME Content-Disposition parameter value that must match corresponding IN-DPARAMETER-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-Disposition: parameter list.
IN-PARAMETER-DEFAULT-*n*	Inputs 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.
IN-PARAMETER-NAME-*n*	Inputs MIME Content-Type parameter name whose value is to be checked; *n* = 0, 1, 2, ...
IN-PARAMETER-VALUE-*n*	Inputs 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	Inputs Subject from enclosing MESSAGE/RFC822 part.
IN-SUBTYPE	Inputs 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	Inputs MIME type to match for conversion (wildcards allowed). The conversion specified is performed only 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	Outputs A1-Format.
OUT-A1-TYPE	Outputs A1-Type.

TABLE 2-16 Conversion Parameters *(Continued)*

Parameter	Description
OUT-CHAN	Outputs 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	Outputs MIME Content-Description if it is different than the input MIME Content-Description.
OUT-DISPOSITION	Outputs MIME Content-Disposition if it is different than the input MIME Content-Disposition.
OUT-DPARAMETER-NAME-*n*	Outputs MIME Content-Disposition parameter name; *n*=0, 1, 2, ...
OUT-DPARAMETER-VALUE-*n*	Outputs MIME Content-Disposition parameter value corresponding to OUT-DPARAMETER-NAME-*n*.
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*	Outputs MIME Content-Type parameter name; *n* = 0, 1, 2, ...
OUT-PARAMETER-VALUE-*n*	Outputs MIME Content-Type parameter value corresponding to OUT-PARAMETER-NAME-*n*.
OUT-SUBTYPE	Outputs MIME type if it is different than the input MIME type.
OUT-TYPE	Outputs 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.

TABLE 2-16 Conversion Parameters *(Continued)*

Parameter	Description
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 2-17 shows the basic set of environment variables available for use by the conversion command.

TABLE 2-17 Environment Variables used by Conversion Channel

Environment Variable	Description
INPUT_TYPE	Content type of the input message part.
INPUT_SUBTYPE	Content subtype of the input message part.
INPUT_DESCRIPTION	Content description of the input message part.
INPUT_DISPOSITION	Content disposition of the input message part.
OUTPUT_FILE	Name of the file where the converter should store its output. The converter should create and write this file.
OUTPUT_FILE	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 a Mapping Table from a Conversion Entry

The value for a conversion parameter may be obtained by calling out a mapping table. The syntax for calling out 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

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 using 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. The 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.

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
```

Use the sample entry shown 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 can specify other values according to the needs of your site. 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.

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.

Note – The mapping file is used for reverse mapping, forward mapping, access control mapping, conversion mapping, and so forth. Additional mapping file information is available in Chapter 6, “IMTA Security and Unsolicited Bulk Email (UBE) Handling,” of the *SIMS Administrator’s Guide*.

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 in the Mapping File

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 consume 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 (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 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.

TABLE 2-18 Mapping Pattern Wildcards

Wildcard	Description
%	Match exactly one character.
*	Match zero or more characters, with maximal or “greedy” left-to-right matching
Back match	**Description**
$ n*	Match the nth wildcard or glob.
Modifiers	Description.
$_	Use minimal or “lazy” left-to-right matching.
$@	Turn off “saving” of the succeeding wildcard or glob.
$^	Turn on “saving” of the succeeding wildcard or glob; this is the default.
Global wildcard	**Description**
$A%	Match one alphabetic character, A--Z or a--z.
$A*	Match zero or more alphabetic characters, A--Z or a--z.
$B%	Match one binary digit (0 or 1).
$B*	Match zero or more binary digits (0 or 1).
$D%	Match one decimal digit 0--9.
$D*	Match zero or more decimal digits 0--9.
$H%	Match one hexadecimal digit 0--9 or A--F.
$H*	Match zero or more hexadecimal digits 0--9 or A--F.
$O%	Match one octal digit 0--7.
$O*	Match zero or more octal digits 0--7.
$S%	Match one symbol set character, for example, 0--9, A--Z, a--z, _, $.
$S*	Match zero or more symbol set characters, for example, 0--9, A--Z, a--z, _, $.
$T%	Match one tab or vertical tab or space character.
$T*	Match zero or more tab or vertical tab or space characters.
$X%	A synonym for $H%.
$X*	A synonym for $H*.
$[c]%	Match character c.

TABLE 2-18 Mapping Pattern Wildcards *(Continued)*

$[c]*	Match arbitrary occurrences of character c.
$[c 1 c 2 ... c n]%	Match exactly one occurrence of character c 1, c 2, or c n.
$[c 1 c 2 ... c n]*	Match arbitrary occurrences of any characters c 1, c 2, or c n.
$[c 1 -c n]%	Match any one character in the range c 1 to c n.
$[c 1 -c n]*	Match arbitrary occurrences of characters in the range c 1 to c n.
$< IPv4>	Match an IPv4 address.

Within globs, that is, within a `$[...]` construct, the backslash character, ,is the quote character. To represent a literal hyphen, `-`, or right bracket, `]`, within a glob the hyphen or right bracket must be quoted with a backslash.

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.

To specify multiple modifiers, or to specify modifiers and a back match, the syntax uses just one dollar character. For instance, to back match the initial wild card, without saving the back match itself, one would use $@0, not $@$0.

Note that the `imta test -mapping` utility may be used to test mapping patterns and specifically to test wildcard behavior in patterns.

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`.

IPv4 Matching

With IPv4 matching, an IP address or subnet is specified, optionally followed by a slash and the number of bits to ignore when checking for a match. For instance,

```
$<123.45.67.0/8>
```

will match anything in the 123.45.67.0 subnet. Or another example is that

```
$<123.45.67.4/2>
```

will match anything in the range 123.45.67.4--123.45.67.7.

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 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 2-19 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 and control 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 2-19 Mapping Template Substitutions and Metacharacters

Substitution sequence	Substitutes
`$n`	The *n*th wildcarded field as counted from left to right starting from 0.
`$#...#`	Sequence number substitution.
`$|...|`	Applies specified mapping table to supplied string.
`${...}`	General database substitution.
`$[...]`	Invokes site-supplied routine; substitute in result.

TABLE 2-19 Mapping Template Substitutions and Metacharacters *(Continued)*

Substitution sequence	Substitutes
Metacharacter	**Description**
$C	Continues the mapping process starting with the next table entry; uses the output string of this entry as the new input string for the mapping process.
$E	Ends the mapping process now; uses the output string from this entry as the final result of the mapping process.
$L	Continues 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, makes one more pass, starting with the first table entry. A subsequent match may override this condition with a $C, $E, or $R metacharacter.
$R	Continues the mapping process starting with the first entry of the mapping table; uses the output string of this entry as the new input string for the mapping process.
$?x?	Mapping entry succeeds x percent of the time.
$\	Forces subsequent text to lowercase.
$^	Forces subsequent text to uppercase.
$_	Leaves 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 ($\, $^, $_*)*

The metacharacter $\ 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. The metacharacter:

- $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. 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*?*)*

The metacharacters $?x? in a mapping table entry cause 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.)The *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 email 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, 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 correct 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 using 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 want 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.

When you want 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)
```

The `argument` and `result` are 252-byte long character string buffers. The `argument` and `result` are passed as a pointer to a character string (for example, in C, as `char*`). The `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.

If you want to use processing control metacharacters such as `$C`, `$R`, or `$L` in a mapping table entry that does a site-supplied routine substitution, you place the processing control metacharacter 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.

Address-Reversal Database, REVERSE Mapping and FORWARD Mapping

Address reversal is the operation consisting of converting an address from an internal form to a public, advertised form. For example, while uid@mailhost.alpha.com might be a valid address within the alpha.com domain, it might not be an appropriate address for the outside world to see. first.last@alpha.com is a more likely public address.

The address reversal operation applies by default to envelop From and all header addresses. This can be changed by setting the value of the REVERSE_ENVELOPE and system options. Address reversal can be turned on or off on a per-channel basis using the reverse channel keyword.

The public address for each user is specified by the mail attribute of the user entry in the directory. The same is true for distribution lists.

The reverse database contains a mapping between any valid address and this public address. It is updated and created by imta dirsync.

The reverse database is created each time you run the imta dirsync command.

The reverse 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 an address is found in the database, the corresponding right 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. The src_channel| destination and channel| internal addresses need to be mapped to *|tcp_local|*@*.stream.com and $|@stream.com$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).

TABLE 2-20 REVERSE mapping table flags

Flags	Description
$Y	Use output as new address.
$N	Address remains unchanged.
$D	Run output through the reversal database.
$A	Add pattern as reverse database entry.
$F	Add pattern as forward database entry.
Flag comparison	**Description**
$:B	Match only header (body) addresses.
$:E	Match only envelope addresses.
$:F	Match only forward pointing addresses.
$:R	Match only backwards pointing addresses.
$:I	Match only message-ids.

As an example, suppose that the internal addresses at `stream.com` are actually of the form `user@host.stream.com`, but, unfortunately, the user name space is such that `user@hosta.stream.com` and `user@hostb.stream.com` specify the same person for all hosts at `stream.com`. Then the following, very simple REVERSE mapping may be used in conjunction with the address-reversal database:

```
REVERSE
* @ *.stream.com            $0@host.stream.com$Y$D
```

This mapping maps addresses of the form `user@anyhost.stream.com` to `user@host.stream.com`. The `$D` metacharacter causes the address-reversal database to be consulted. The address-reversal database should contain entries of the form:

```
user@host.stream.com     first.last@stream.com
```

The reverse and noreverse channel keywords, and the IMTA options USE_REVERSE_DATABASE and REVERSE_ENVELOPE might be 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. These 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 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. Circumstances can 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.stream.com` associated with the `native` channel produces RFC 822 addresses of the general form:

```
"lastname, firstname"@am.sigurd.stream.com
```

or

```
"lastname,firstname"@am.sigurd.stream.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.stream.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_gateway 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 results. The REVERSE mapping shown assumes that bit 3 in the IMTA option USE_REVERSE_DATABASE is set.

```
REVERSE
   *|mr_gateway|"*,$ *"@am.sigurd.stream.com $Y"$1,$ $2"@am.sigurd.nocompany.com
   *|mr_gateway|"*,*"@am.sigurd.stream.com   $Y"$1,$ $2"@am.sigurd.nocompany.com
   *|*|"*,$ *"@am.sigurd.stream.com        $Y$3.$2@am.sigurd.nocompany.com
   *|*|"*,*"@am.sigurd.stream.com          $Y$3.$2@am.sigurd.nocompany.com
   *|mr_gateway|*.*@am.sigurd.stream.com     $Y"$2,$ $1"@am.sigurd.nocompany.com
   *|*|*.*@am.sigurd.stream.com            $Y$2.$3@am.sigurd.nocompany.com

FORWARD
   "*,$ *"@am.sigurd.stream.com            $Y"$0,$ $1"@am.sigurd.nocompany.com
   "*,*"@am.sigurd.stream.com              $Y"$0,$ $1"@am.sigurd.nocompany.com
   *.*@am.sigurd.stream.com                $Y"$1,$ $0"@am.sigurd.nocompany.com
```

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 IMTA 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
```

The *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 2-21.

TABLE 2-21 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 an "illegal host or domain" error. This is the same error that would occur if the address were simply illegal. Although confusing, this usage 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)	Sets the size of the alias hash table. This is an upper limit on the number of aliases that can be defined in the alias file. The default is 256; the maximum value is 32,767.
`ALIAS_MEMBER_SIZE` (Integer <= 20,000)	Controls the size of the index table that contains the list of alias translation value pointers. The total number of addresses on the right sides of all of the alias definitions in the alias file cannot exceed this value. The default is 320; the maximum value is 20,000.

TABLE 2-21 Option File Options *(Continued)*

Options	Description
BLOCK_LIMIT (Integer > 0)	Places an absolute limit on the size, in blocks, of any message that may be sent or received with IMTA. Any message exceeding this size will be rejected. By default, IMTA imposes no size limits. Note 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 using 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.
BOUNCE_BLOCK_LIMIT	Used to force bounces of messages over the specified size to return only the message headers, rather than the full message content.
CHANNEL_TABLE_SIZE (Integer <= 32,767)	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.
COMMENT_CHARS	Sets the comment characters in IMTA configuration files.
CONVERSION_SIZE (Integer <= 2000)	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)	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 of 0 disables this output.
DOMAIN_HASH_SIZE (Integer <= 32,767)	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 is 32,767.
EXPROUTE_FORWARD (Integer 0 or 1)	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)	Controls how many delivery attempt history records are included in returned messages. The delivery history provides an indication of how many delivery attempts were made and might indicate the reason the delivery attempts failed. The default value for this option is 20.
HELD_SND_OPR	Controls the production of operator messages when a message is forced into a held state because it has too many Received: header lines.

TABLE 2-21 Option File Options *(Continued)*

Options	Description
HOST_HASH_SIZE (Integer <= 32,767)	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)	Specifies the domain name to use when constructing message IDs. By default, the official host name of the local channel is used.
IMPROUTE_FORWARD (Integer 0 or 1)	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)	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.The linelimit channel keyword can be used to impose limits on a per channel basis.
LINES_TO_RETURN (Integer)	Controls how many lines of message content IMTA includes when bouncing messages. The default is 20.
LOG_CONNECTION (0 or 1)	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_DELAY_BUG	Specifies the bins for delivery delay range counters.
LOG_FILENAME (0 or 1)	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)	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 2-21 Option File Options *(Continued)*

Options	Description
LOG_HEADER (0 or 1)	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)	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)	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_PROCESS	Includes the enqueuing process ID in IMTA log entries.
LOG_SNDOPR	Controls the production of operator (OPCOM) messages or syslog messages by the IMTA message logging facility.
LOG_SIZE_BINS	Specifies the bins for message size range counters.
LOG_USERNAME (0 or 1)	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)	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)	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)	Controls what fraction of the available message blocks can be used by message headers.
MAX_HEADER_LINE_USE (Real Number Between 0 and 1)	Controls what fraction of the available message lines can be used by message headers.

TABLE 2-21 Option File Options *(Continued)*

Options	Description
MAX_INTERNAL_BLOCKS (Integer)	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_MIME_LEVELS	Specify the maximum depth to which IMTA should process MIME messages. The default is 100, which means that IMTA will process up to 100 levels of message nesting.
MAX_MIME_PARTS	Specify the maximum number of MIME parts that IMTA should process in a MIME message.
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.
MISSING_RECIPIENT_POLICY	Legalizes messages that lack any recipient headers.
NORMAL_BLOCK_LIMIT (Integer)	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.
NON_URGENT_BLOCK_LIMIT (Integer)	Used to instruct IMTA to downgrade the priority of messages based on size: Messages above the specified size will be downgraded to lower than nonurgent priority; 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.
POST_DEBUG (0 or 1)	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.

TABLE 2-21 Option File Options *(Continued)*

Options	Description
RECEIVED_DOMAIN (String)	Sets the domain name to use when constructing Received headers. By default, the official host name of the local channel.
RETURN_ADDRESS (String)	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 pileups of huge numbers of spurious error messages.
RETURN_DEBUG (0 or 1)	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.
RETURN_DELIVERY_HISTORY (0 or 1)	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)	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 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 that the returnenvelope channel keyword can be used to impose this sort of control on a per-channel basis.
RETURN_PERSONAL (String)	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)	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.

TABLE 2-21 Option File Options *(Continued)*

Options	Description
SEPARATE_CONNECTION_LOG (0 or 1)	Controls whether the connection log information generated by setting LOG_CONNECTION =1 is stored in the usual IMTA message logging files, mail.log* or is stored separately in connection.log* files. The default (0) causes connection logging to be stored in the regular message log files; 1 causes the connection logging to be stored separately.
STRING_POOL_SIZE (Integer <= 10,000,000)	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.
URGENT_BLOCK_LIMIT (Integer)	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)	Controls whether IMTA uses the alias database as a source of system aliases for local addresses. The default (1), means that IMTA will check the database if it exists. A value of 0 will disable this use of the alias database.
USE_DOMAIN_DATABASE	Controls the use of the domain database. The default (1) means that IMTA will check the database if it exists. 0
USE_ERRORS_TO (0 or 1)	Controls whether IMTA uses 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. The default (0), disable uses of this header line.
USE_FORWARD_DATABASE	Control use of the forward database.
USE_REVERSE_DATABASE (0-31)	Controls whether IMTA uses 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 2-22.
USE_WARNINGS_TO (0 or 1)	Controls whether IMTA uses 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.
WILD_POOL_SIZE (integer)	Controls the total number of patterns that appear throughout mapping tables. the default is 8000. The maximum allowed is 200,000.

TABLE 2-22 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. 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. The use of `defaults` guards against the inevitable expansion of IMTA's known header line table in future releases.

Various options can then be specified to control the retention of the corresponding header lines. The available options are listed in TABLE 2-23.

TABLE 2-23 Header options

Option	Description
ADD (Quoted String)	Creates a 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)	Creates a new header line of the given type 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)	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)	Controls the maximum number of characters that can 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)	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 can consume. A value of -1 is interpreted as a request to suppress this header line type completely.
MAXLINES (Integer)	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.

TABLE 2-23 Header options *(Continued)*

Option	Description
PRECEDENCE (Integer)	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. 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.

RELABEL (header name)	Changes a header line to another header line; that is, the name of the header is changed, but the value remains the same. For instance,
	`X-MSMail-Priority: RELABEL="Priority"`
	`X-Priority: RELABEL="Importance"`

Tailor File

The IMTA tailor file (`imta_tailor`) 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 – Do not edit this file unless absolutely necessary.

An option setting has the form:

```
option=value
```

The *value* can 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 shown in TABLE 2-24.

TABLE 2-24 tailor File Options

Option	Description
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.properties.
IMTA_ALIAS_DATABASE	IMTA alias database. The default is /var/opt/SUNWmail/imta/db/aliasesdb.
IMTA_ALIAS_FILE	IMTA aliases file. Aliases not set in the directory, for example, postmaster, are set in this file. The default is /etc/opt/SUNWmail/imta/aliases.
IMTA_CHARSET_DATA	Specifies where IMTA compiled character set data is located. The default 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 shell scripts are located. The default is /opt/SUNWmail/imta/lib/.
IMTA_CONFIG_DATA	Compiled configuration for the IMTA. The default is /opt/SUNWmail/imta/lib/config_data.
IMTA_CONFIG_FILE	IMTA configuration file. 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 is /etc/opt/SUNWmail/imta/conversions.
IMTA_DB_HASH_SIZE	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	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 is 10 and works fine for databases with up to 4 million entries.
IMTA_DISPATCHER_CONFIG	IMTA dispatcher's configuration file. The default is /etc/opt/SUNWmail/imta/dispatcher.cnf.
IMTA_DOMAIN_DATABASE	Database used to store additional rewrite rules. The default is /var/opt/SUNWmail/imta/db/domaindb
IMTA_DNSRULES	IMTA DNS configuration library. The default is /opt/SUNWmail/imta/lib/imdnsrules.so.
IMTA_FORWARD_DATABASE	Not used for SIMS 4.0.
IMTA_GENERAL_DATABASE	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.

TABLE 2-24 tailor File Options *(Continued)*

Option	Description
IMTA_HELP	Location of the help files for the imta utility. The default is /opt/SUNWmail/imta/lib.
IMTA_JBC_CONFIG_FILE	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	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 that 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 utility library. By default it is /opt/SUNWmail/lib/libimtautil.so.1.
IMTA_LOG	Location of the IMTA log files. The default is /var/opt/SUNWmail/imta/log/.
IMTA_MAPPING_FILE	File used for setting access control rules, reverse mapping rules, forward mapping rules, and so forth. 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	IMTA message queue directory. The default is /var/opt/SUNWmail/imta/queue.
IMTA_QUEUE_CACHE_DATABASE	Location of the IMTA 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.

TABLE 2-24 tailor File Options *(Continued)*

Option	Description
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	IMTA 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 its 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 option 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 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 the /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	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	Can perform certain privileged operations as a member of this group. The default is mail.

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 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
```

The *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 as follows:

TABLE 2-25 dirsync File Options

Option	Description
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/.
USER_SPEC_INTERNAL	Used to create aliases and domain rewrite rules for hosted domains (%u?%d is the default). Where %u is replaced by the user part and %d is replaced by the domain part.
USER_SPEC	Used to create addresses for a channel for which no spec has been specified in the channel option file. (This does not apply to the default channels.)

Autoreply Option File

This file is used for setting options for the autoreply or vacation program. This file should be located in the IMTA 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
```

The *value* may be either a string or an integer, depending on the option's requirements.

The available options are:

TABLE 2-26 autoreply File Options

Option	Description
`DEBUG`	Determines whether a trace file is created for each autoreply. The default is 0 and this facility is off. A value of 1 creates an autoreply trace file for each autoreply sent in the IMTA 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, if this option is not set, is 168 (for example, once a week).

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`) can 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 is 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 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.

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 wildcard 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 wildcard, it will match any channel whose name begins with tcp_.

TABLE 2-27 shows the available options.

TABLE 2-27 Job Controller Configuration File Options

Option	Description
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 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 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.

TABLE 2-27 Job Controller Configuration File Options *(Continued)*

Option	Description
POLL_RUNS_SLAVE=0 or 1	Controls whether jobs submitted with the poll parameter execute both master and slave directions of a channel, or whether poll jobs execute only the master direction of a channel. POLL_RUNS_SLAVE=1 is the default and should be used for most channels. POLL_RUNS_SLAVE=0 should be specified for IMTA-DIRSYNC channels.
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 unless the default conflicts with another UDP application on your system. If you do change this option, 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 is the command that stops 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
```

The *path* is the full path name 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.

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, control the number of SMTP server processes and the number of connections each SMTP server process handles.

Creation and Expiration of SMTP Server Processes

Automatic housekeeping facilities within the SMTP Dispatcher control the creation of new and expiration of old or idle SMTP server processes. The basic options that control the SMTP Dispatcher's behavior 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.

It is possible that a currently running SMTP server process might not be able to accept any connections 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.

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
```

The *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, using lines of the following form:

```
SERVICE=service-name
```

The *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/lib/tcp_smtp_server
LOGFILE=/var/opt/SUNWmail/imta/log/tcp_smtp_server.log
```

TABLE 2-28 shows the available options.

TABLE 2-28 Dispatcher configuration file options

Option	Description
`BACKLOG=`*integer*	Controls the depth of the TCP backlog queue for the socket. The default value for each service is `MAX_CONNS*MAX_PROCS` (with a minimum value of 5). This option should not be set higher than the underlying TCP/IP kernel supports.

`ENABLE_RBL=`*0 or 1*	Specifying `ENABLE_RBL=1` causes the Dispatcher to compare incoming connections to the “Black Hole” list at maps.vix.com. For instance, if the Dispatcher receives a connection from 192.168.51.32, then it will attempt to obtain the IP address for the hostname 32.51.168.192.rbl.maps.vix.com. If the query is successful, the connection will be closed rather than handed off to a worker process. If this option is enabled on a well-known port (25, 110, or 143), then a standard message such as the one below will be sent before the connection is closed:
	
	`5.7.1 Mail from 192.168.51.32 refused, see http://maps.vix.com/rbl/`
	
	If you want the IMTA to log such rejections, set the 24th bit of the Dispatcher debugging `DEBUG` option, `DEBUG=16%1000000`, to cause logging of the rejections to the `dispatcher.log` file; entries will take the form:
	
	`access_control: host a.b.c.d found on RBL list and rejected`

TABLE 2-28 Dispatcher configuration file options *(Continued)*

Option	Description
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.
INTERFACE_ADDRESS=*IP address*	The INTERFACE_ADDRESS option can be used to specify the IP address interface to which the Dispatcher service should bind. By default, the Dispatcher binds to all IP addresses. But for systems having multiple network interfaces each with its own IP address, it may be useful to bind different services to the different interfaces. Note that if INTERFACE_ADDRESS is specified for a service, then that is the only interface IP address to which that Dispatcher service will bind. Only one such explicit interface IP address may be specified for a particular service (though other similar Dispatcher services may be defined for other interface IP addresses).
IDENT=*0 or 1*	If IDENT=1 is set for a service, it causes the Dispatcher to try an IDENT query on incoming connections for that service, and to note the remote username (if available) as part of the Dispatcher statistics. The default is IDENT=0, meaning that no such query is made.
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_HANDOFFS=*integer*	Specifies the maximum number of concurrent asynchronous handoffs in progress that the Dispatcher will allow for newly established TCP/IP connections to a service port. The default value is 5.
MAX_IDLE_TIME=*integer*	Specifies the maximum idle time for a SMTP server process. When an SMTP server process has had no active connections for this period, it 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 an 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 an 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.

TABLE 2-28 Dispatcher configuration file options *(Continued)*

Option	Description
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.

PARAMETER	The interpretation and allowed values for the PARAMETER option are service specific. In the case of an SMTP service, the PARAMETER option may be set to CHANNEL=channelname, to associate a default TCP/IP channel with the port for that SMTP service. For instance,
	`[SERVICE=SMTP]`
	`PORT=25`
	`...`
	`PARAMETER=CHANNEL=tcp_incoming`
	This can be useful if you want to run SMTP servers on multiple ports -- perhaps because your internal POP and IMAP clients have been configured to use a port other than the normal port 25, thus separating their message traffic from incoming SMTP messages from external SMTP hosts---and if you want to associate different TCP/IP channels with the different port numbers.

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 farther 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.

Debugging and Log Files

Service Dispatcher error and debugging output (if enabled) are written to the file `dispatcher.log` in the IMTA log directory.

Debugging output may be enabled using the option DEBUG in the Dispatcher configuration file, or on a per-process level, using the `IMTA_DISPATCHER_DEBUG` environment variable (UNIX).

The DEBUG option or IMTA_DISPATCHER_DEBUG environment variable (UNIX) defines a 32-bit debug mask in hexadecimal. Enabling all debugging is done by setting the option to -1, or by defining the logical or environment variable system-wide to the value FFFFFFFF. The actual meaning of each bit is described in TABLE 2-29.

TABLE 2-29 Dispatcher Debugging Bits

Bit	Hexadecimal value	Decimal value	Usage
0	x 00001	1	Basic Service Dispatcher main module debugging.
1	x 00002	2	Extra Service Dispatcher main module debugging.
2	x 00004	4	Service Dispatcher configuration file logging.
3	x 00008	8	Basic Service Dispatcher miscellaneous debugging.
4	x 00010	16	Basic service debugging.
5	x 00020	32	Extra service debugging.
6	x 00040	64	Process related service debugging.
7	x 00080	128	Not used.
8	x 00100	256	Basic Service Dispatcher and process communication debugging.
9	x 00200	512	Extra Service Dispatcher and process communication debugging.
10	x 00400	1024	Packet level communication debugging.
11	x 00800	2048	Not used.
12	x 01000	4096	Basic Worker Process debugging.
13	x 02000	8192	Extra Worker Process debugging.
14	x 04000	16384	Additional Worker Process debugging, particularly connection handoffs.
15	x 08000	32768	Not used.
16	x 10000	65536	Basic Worker Process to Service Dispatcher I/O debugging.
17	x 20000	131072	Extra Worker Process to Service Dispatcher I/O debugging.
20	x 100000	1048576	Basic statistics debugging.
21	x 200000	2097152	Extra statistics debugging.
24	x 1000000	16777216	Log PORT_ACCESS denials to the dispatcher.log file.

System Parameters on Solaris

The system's heap size (`datasize`) must be enough to accommodate the Dispatcher's thread stack usage. For each Dispatcher service compute `STACKSIZE*MAX_CONNS`, and then add up the values computed for each service. The system's heap size needs to be at least twice this number.

To display the heap size (that is, default `datasize`), use the `csh` command

```
# limit
```

or the `ksh` command

```
# ulimit -a
```

or the utility

```
# sysdef
```

CHAPTER 3

Sun Directory Services Directory Information Tree and Schema

Introduction

This chapter describes a schema for Internet mail routing and delivery, but the core of this schema also serves as the core for other Internet services. It also describes the SIMS Directory Information Tree (DIT) and Schema. You use the DIT and schema information for provisioning and debugging.

The examples provided throughout the document show how the schema is suited for Internet mail and also illustrate the modularity that provides support for service- and vendor-independent interoperability.

For information on how Sun Directory Services relates to SIMS refer to the *SIMS Concepts Guide* and *SIMS System Administration Guide*. For detailed information on administration and usage, refer to the *Sun Directory Services* documentation.

The goal of this document is to precisely identify the format, type, and acceptable values of the directory entries used by Sun Internet Mail Server. This document has two intended audiences: engineering groups and customers who develop their own tools for populating the directory with data.

This document assumes that the reader is familiar and comfortable with installing and managing SIMS. Readers should be familiar with LDAP Directory Interchange Format (ldif(1)). It is also assumed that the reader has read and understands the following documents:

- X.520 (1993)
- X.520 (1998)
- X.521 (1993)
- RFC 2256
- http://www.ietf.org/Internet-drafts/draft-smith-ldap-inetorgperson-00.txt

Producers and Consumers of the Mail Schema

A producer is defined as any software component that can create, or subsequently modify a value for an attribute in an object class. A consumer is defined as any software component that retrieves and uses attributes in the process of accomplishing some task.

The following sections defining the LDAP mail schema specify the producer(s) and consumer(s) of each of the attributes. Here, an Internet mail system is subdivided into the following components:

Message Transfer Agent (mta) – Communicates through Simple Mail Transfer Protocol [SMTP] and is responsible for either routing mail to another MTA or delivering the message into a mailbox.

Message Store / Message Access agent (msma) – Responsible for supporting access by email client software to a user's mail folders. This component may be:

- a traditional Message Store Agent, with local storage of user's mail folders
- a proxy server between the email client and another MSMA agent
- a referral agent that returns the name of another MSMA agent to the email client
- a combination of all three of the above.

In proxy mode, the agent can be implemented as a protocol router for **POP** [POP] and/or **IMAP** [IMAP] requests. When functioning as an end-point for mail access requests, the MSMA agent will retrieve and delete messages, and otherwise manipulate the folders belonging to the user in the message store.

client – Any software agent producing and/or consuming mail directory entries and interpreting the semantics of object class attributes as specified here. These are usually user agents acting on behalf of a non-privileged end-user, and can range from stand-alone email clients to cgi-bin scripts or Java servlets invoked by a web server in response to HTTP commands from a user's web browser.

admin – Software agents that provision the directory (creation, update of entries). This class of producer/consumer is usually acting on behalf of a privileged user (for example, a site administrator). Such agents can range from GUI stand-alone or web-based administration consoles, to character-based scripts invoking low-level LDAP commands. The heading for each of the attribute sections lists the following:

`(<matching rule>, <multi-valued>, {<producer/consumer>})`

Where:

`<matching rule>` **–** Matching rule for this attribute. For example, cis, ces,...

`<multi-valued>` **–** Number of attributes allowed per entry. For example, 1, 0-1, 0-many, ...

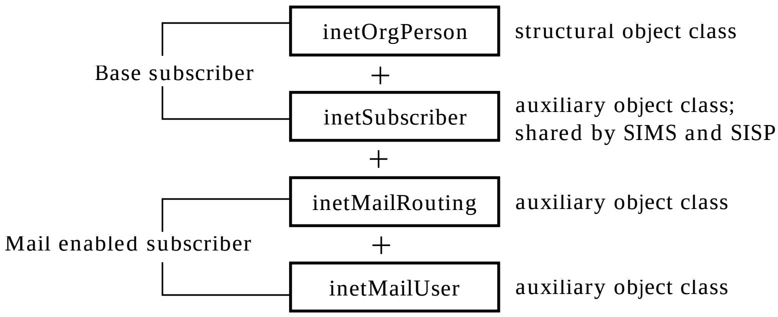
`<producer/consumer>` **–** A comma-separated list of the producers and/or consumers of this attribute from the list of Internet mail system components above.

Directory Schema and DIT Specification

Structural object classes are used to define nodes in the directory. Auxiliary object classes can be used to extend the set of attributes that a directory entry might contain. The data model used by SIMS is to extend entries, defined by standard object classes, by overlaying them with service-specific auxiliary object classes.

The shared auxiliary object classes hold a minimum of attributes and can be used to define generic entries in the directory (for example, `inetSubscriber` is used to define a basic user entry). Service-specific attributes are encapsulated in the auxiliary object classes. To enable a user/subscriber to start using a service or to enable a host object to hold service-specific attributes, these auxiliary object classes are used to extend the generic entries.

The sections that follow define the object classes used to define the directory entries (user, groups, hosted domains, host objects, and so on) and the specifications of the attributes in these object classes.



Directory Information Tree

The role of a directory service is to support the storage and retrieval of data. Visualize the entries in an LDAP directory as a tree-like structure. This mirrors the tree model used by most file systems and is referred to as the directory information tree (DIT).

Just as a file path uniquely identifies a file within a file system, a directory entry is uniquely identified within the DIT using a distinguished name (DN). A DN identifies the entry with a comma-separated list of attribute and attribute value pair. The DN's left-most attribute-value pair is known as the relative distinguished name (RDN).

Following the RDN, each successive attribute-value pair is the RDN of the next parent node in the DIT hierarchy, each one representing a potential branch point above the entry. The final, or right-most, attribute-value pair represents the conceptual root point of the DIT.

SIMS 4.0 allows the data to be represented in two ways in the directory. The recommended approach is to have a single domain component tree (DC tree) that contains all the data used by their services: users, distribution lists, hosted domains entries, and so on. Starting with this release of SIMS, DC tree deployments will be recommended and the installer will always create the DC tree for all new installs.

For sites that need to maintain compatibility the OSI style tree, or that don't want to migrate the directory of SIMS 3.0 to conform with the recommended DIT, an alternate data layout is supported. This alternate layout uses a combination of primary and secondary tree.

The primary tree is the repository of all users and distribution list data and is patterned after an OSI DIT.

The secondary tree is a Domain Component tree (DC tree) that mirrors the DNS hierarchy. The secondary tree also holds the virtual domains, because domain data logically belongs in the DC tree.

The DC tree provides the mapping from the DNS namespace to the primary namespace where all users and distribution lists are defined. This mapping is used by message transfer agents for building routing tables and in making message routing decisions; it is also used by the IMAP/POP servers when authenticating users.

The root entry of the DIT is defined by the suffix value of the directory server. Therefore, the LDAP directory server will have to support multiple suffixes for multiple DITs to be created. Sun Directory Server and Netscape Directory Server support multiple DITs.

OSI trees with a shadow DC tree are deprecated with an eye towards discontinuing support for such deployments in the future.

As noted, SIMS 4.0 supports user/distribution list data in a DC tree or a combination of DC and OSI trees. Examples using both are shown below.

Data In a Single Domain Component Tree

The recommended approach for setting up the directory information tree in a directory server is to create all entries that SIMS depends on in a tree that is patterned after a domain component tree. The tree should be rooted at `o=Internet`. Configure the directory server with this suffix.

All nodes directly below `o=internet` correspond to the top-level domains in the DNS namespace. For example, some of the nodes below the root would be `dc=com,o=internet`; `dc=fr,o=internet; dc=edu,o=internet`; and so on.

These top-level nodes that correspond to the top-level domains contain the node for organizations. Examples of these are `dc=sun,dc=com,o=internet` and `dc=sfr,dc=fr,o=internet`. DNS domains within an organization's top level DNS domain are represented by corresponding containers of the format `dc=<sub_domain>,dc=sun,dc=com,o=internet`. Each node representing the organization or sub-domain in the organization is required to have the following organizational units:

- organizational unit: people – User entries are defined so that they are contained within the people organizational unit.
- organizational unit: groups – Distribution list entries are defined so that they are contained within the groups organizational unit.
- organizational unit: services – Entries for services are contained within the services organizational unit.

In the figure below, the DN for a user entry in engineering organizational unit will have a suffix of ou=people,dc=engineering,dc=sun,dc=com,o=internet, preceded by the entry's RDN. For example:

cn=John Doe,ou=people,dc=engineering,dc=sun,dc=com,o=internet.

Each containers — o=internet; dc=<top_level_dns_domain>,o=internet; dc=<dns_suffix_for_org>, dc=<top_level_dns_domain>,o=internet — and then the organizational units people, groups, and services are directory entries themselves and are made up of the following object classes.

DN OF NODE IN DIT	OBJECT CLASSES ASSOCIATED WITH DIRECTORY ENTRY
o=internet	organization
dc=<top_level_dns_domain>, o=internet	domain
dc=<dns_suffix_for_org>,dc=<top_level_dns_domain>, o=internet	domain simsDomain inetDomain
dc=<sub_domain>,dc=<dns_suffix_for_org>,dc=<top_level_dns_domain>, o=internet	domain simsDomain inetDomain
ou=people, dc=<dns_suffix_for_org>,dc=<top_level_dns_domain>, o=internet	organizationalUnit
ou=groups, dc=<dns_suffix_for_org>,dc=<top_level_dns_domain>, o=internet	organizationalUnit
ou=services, dc=<dns_suffix_for_org>,dc=<top_level_dns_domain>, o=internet	organizationalUnit

The directory entry for the root node is represented by the following LDIF:

```
dn: o=internet
organization: internet
objectclass: organization
description: Root node of the Domain Component (DC) Tree
```

The directory entry for the second level node is represented by the following LDIF:

```
dn: dc=com, o=internet
domainComponent: com
objectclass: domain
description: Top-level node for COM domains.
```

The directory entry for the third level node is represented by the following LDIF:

```
dn: dc=sun, dc=com, o=internet
domainComponent: sun
objectclass: domain
objectclass: inetDomain
objectclass: simsDomain
inetTreeStyle: DC
description: Top level node of sun.com
dnsDomainName: sun.com
```

The directory entry for people container is represented by the following LDIF (groups and services follow the same format).

```
dn: ou=People, dc=sun, dc=com, o=internet
organizationalUnit: people
objectclass: organizationalUnit
```

You can add additional attributes to the above nodes, especially to the nodes defined with the inetDomain object class. Attributes of the inetDomain and SimsDomain object classes are used to set the various properties of a virtual domain.

Data in OSI and DC trees

Primary tree

The primary tree is patterned after an OSI tree and is rooted at c=<country-name>. Therefore, the suffix for the primary tree is set to c=<country_name>. The nodes in bold correspond to a site's organization structure. Each node in the DIT that corresponds to a valid DNS domain or sub-domain in the organization is required to have the following organization units:

- organizational unit: people
- organizational unit: groups
- organizational unit: services

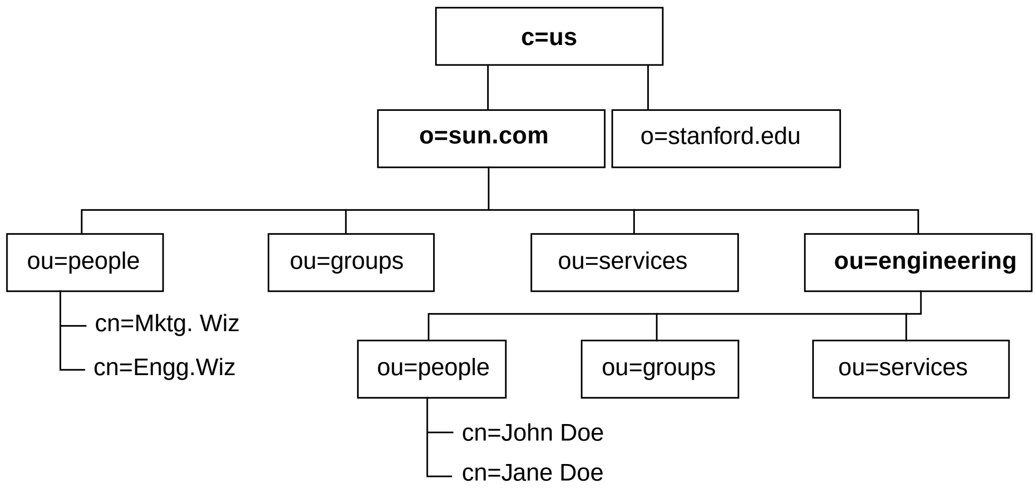


FIGURE 3-1 SIMS OSI (Primary) Directory Information Tree

User entries are defined so that they are contained within the people organizational unit. Distribution list entries are defined so that they are contained within the groups organizational unit. Service state is located within the services organizational unit.

In FIGURE 3-1, the DN for a user entry in the engineering organizational unit will have a suffix of `ou=people,ou=engineering,o=sun.com,c=us`, preceded by the entry's RDN.

Each container is a directory entry and is defined by the `organizationalUnit` object class. The directory entry for people container is shown below (groups and services follow the same format).

```
dn: ou=People,o=sun.com,c=us
organizationalUnit: people
objectclass: organizationalUnit
```

In the illustration above, the root of the DIT is defined by the suffix `c=us`. This directory entry is defined by `country` object class. The directory entry for the root entry is shown below.

```
dn: c=us
country: us
objectclass: country
description: Root node of the OSI tree
```

Second level nodes correspond to the organizations contained in the directory server and are defined by organization object class. An example of the entry for a second level node is shown below.

```
o: sun.com,c=us
o: sun.com
o: Sun Microsystems, Inc
ojectclass: organization
description: OSI node for sun.com
```

The organization nodes are followed by entries for the containers for people (`ou=people`), groups (`ou=groups`), and services (`ou=services`) as described earlier. Organizations that have divisions and subdivisions (for example, engineering and marketing) should represent their organizational hierarchy by creating containers for each division and subdivision. These containers are defined by the `organization` object class. An example of the entry for a division is shown below.

```
dn: ou=engineering,o=sun.com,c=us
ou: Sun Microsystems engineering organization
objectclass: organizationalUnit
```

Note – Each node for an organization or a division that can hold users and groups must have the three containers for people, groups, and services.

Secondary tree

The secondary tree provides the mapping from DNS name space to the OSI name space and follows the recommendations of RFC 1279, section 11. The tree is rooted at `o=internet` and is structured exactly the same way as if the domain component tree was the primary tree—with one major difference. When the DC tree is not the primary tree, the attribute `inetlabeledURI` in the nodes created with `inetDomain` is set to point to the DN of the OSI tree containing the users and groups for that DNS domain (for example, `ldap:///ou=engineering,o=sun.com,c=us??sub`).

Each node corresponding to a DNS domain is created with the following object classes — `top`, `domain`, `inetDomain`, and `simsDomain`. For example, the directory entry for the *eng.sun.com* domain is shown below.

```
dn: dc=eng,dc=sun,dc=com,o=internet
dc: eng
description: DC node for eng.sun.com
objectclass: top
objectclass: domain
objectclass: inetDomain
objectclass: simsDomain
inettreestyle: OSI
inetdomainstatus: active
simsrecursive: 0
simsdomainversion: 1.0
dnsdomainname: eng.sun.com
inetauthorizedservices: imap
netauthorizedservices: pop3
inetauthorizedservices: imaps
inetauthorizedservices: pop3s
inetauthorizedservices: smtp
inetauthorizedservices: sunw_webaccess
inetauthorizedservices: sunw_calendar
inetlabeleduri: ldap:///ou=engineering,o=sun.com,c=us??sub
maxentries: -1
maxmailboxes: -1
maxdistributionlists: -1
mailhosts: mail.eng.sun.com
preferredmailhost: mail.eng.sun.com
```

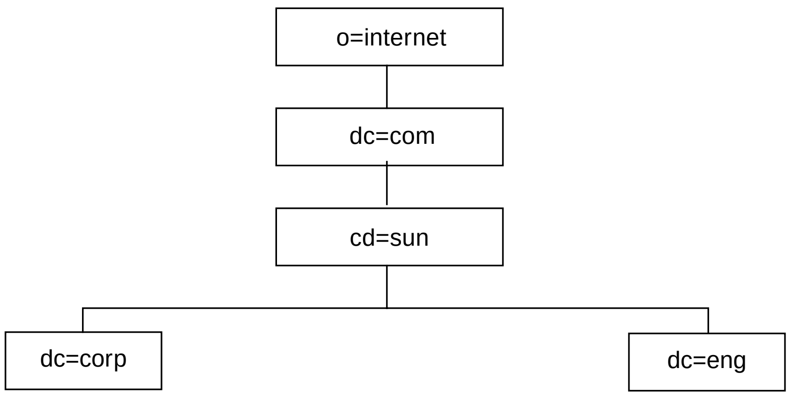


FIGURE 3-2 SIMS Domain Component (Secondary) Directory Information Tree

Note – It is important that the associations between the domain component tree (secondary) and the OSI tree (primary) be set up for applications to search for users and groups within a given DNS domain

FIGURE 3-2 shows that the node `dc=eng,dc=Sun,dc=com,o=Internet` points to the DN `ou=engineering,o=sun.com,c=us`. SIMS looks for users in the *eng.sun.com* DNS domain in the sub-tree `ou=People,ou=engineering,o=sun.com,c=us`.

Clients of the directory should use the `inetTreeStyle` attribute of `inetDomain` object class to determine whether the users, groups, and services are under the DC tree (`inetTreeStyle=DC`) or under the OSI tree (`inetTreeStyle=OSI`). When `inetTreeStyle=OSI`, value of `inetlabeledURI` is used to determine the DN in the OSI tree that holds the user entries.

Attribute Syntax

The description of the attributes includes, among other things, the syntax of the attribute. This syntax is a directive to the Directory Service Agent (DSA). The possible syntaxes are:

- **dn** – A string distinguished name (as defined in rfc1779).
- **cis** – A case ignore string.
- **ces** – A case exact string (case is significant during comparisons).
- **bin** – A binary value.
- **tel** – A string telephone number (blanks and dashes are ignored during comparisons)
- **utctime** – UTC time stamp in the following format YYMMDDHHMMSS.
- **protected** – An encrypted value. In Sun Directory Server 1.0 and 3.1, a value prefixed with {crypt} denotes that it has already been encrypted according to UNIX crypt. A value with no prefix is assumed to be in the clear and is crypted by the directory before it is stored. In SunDS 3.1, a value prefixed with {sunds} denotes that the value is encrypted using a MD5 hashing scheme. Attributes with protected syntax are not returned in searches unless the credentials that the client is using when binding to the directory has the access (see ACL for Sun Directory Server) over the attribute with a protected syntax.

Attributes may appear more than once in a directory entry. For each attribute in the object attribute tables that follow the possible number of attribute values are:

- **1** – one and only one value
- **0-1** – zero or one value

- **0-many** – zero or more values
- **1-many** – one or more values

This is the number of values required by the schema checking process.

SIMS provides a tool — `imldifsync` — that assists in migrating users and distribution lists from NIS and NIS+ into the directory.

The sections that follow describe the object classes, associated attributes, and the directory information tree pertinent to the functioning of services provided by SIMS.

Services and Functions

SIMS provides the following services:

- **ms** – Sun Message Store
- **ma** – IMAP/POP message access
- **mta** – SMTP mail service
- **admin** – SIMS administration service

These services, individually or in combination, provide a set of functionality pertinent to an Internet customer. These are defined below:

- **auth** – user authentication to mailbox and directory data.
- **authorization** – authorization to execute a privileged command, such as delete a mailbox.
- **routing** – routing of messages. Includes routing to the correct mail server and to the correct channel.
- **access** – access control over directory objects

Object class attributes, described in the sections below, are marked with a list of services that depend on that attribute. The format of the notation is described by the following BNF.

```
services            ::= "{" service-name [:service-name] "}"
service-name        ::= service [:service-name]
service             ::="ms"|"ma"|"mta"|"smcs"|"admin"|
                    "spm"|"ftp"|"nntp"|"web"| "sia" |
                    "sism"
```

Object Classes Used by Sun Internet Mail Server 4.0

The following section describes the object classes used to define the directory information tree (DIT), mail users, and distribution lists. Some of the object classes used to create the LDAP entries used by Sun Internet Mail Server 4.0 are defined in X.500 standard or Internet Drafts. SIMS extends the capabilities of those object classes and defines new mail specific object classes and relevant attributes. These are used to overlay basic LDAP entries so that additional mail specific attributes can be stored in them.

In the following sections, all the object classes used by SIMS are described. The first section contains information about the object classes used to create the LDAP entries that make the DIT. It also contains object classes used to make a domain object a virtual domain. This is followed by internet mail users and internet mail groups.

Directory Information Tree and Virtual Domain Object Classes

Users and distribution lists are leaf nodes in the DIT. The DIT is defined by the following object classes.

- `country` – Attributes used to describe a country.
- `organization` – Attributes used to describe an organization.
- `organizationalUnit` – Attributes used to describe an organizational unit container. Used to create the `OU=People`, `OU=Groups`, and `OU=Services` containers, and to create other organizational units contained in the OSI tree.
- `domain` – Attributes used to describe the domain component nodes of the DC tree. These nodes may be containers (domain and sub-domains) and leaf nodes (hosts).
- `inetDomain` – Attributes used to describe the additional properties for a hosted domain. This object class is associated with directory containers that correspond to a DNS domains. In an Internet style DIT, this object class is associated with every DC (domain component) node that represents a DNS domain.
- `simsDomain` – Attributes used to describe the additional properties for an email domain. This object class is associated with directory containers that correspond to a DNS domains. In an Internet style DIT, this object class is associated with every DC (domain component) node that represents a DNS domain.

Note – Attributes listed in the "Required Attribute" tables are used either for the basic functionality of SIMS or by one or more extended services. They are not necessarily required by the schema checking process. Some attributes listed as required can be omitted, and the object will still pass the schema checking process. If the number of values for an attribute is listed as either 1 (only one value) or 1-many (one or more values), then the attribute is required for the object to pass schema checking. If the number of values for an attribute is listed as either 0-1 (zero or one value) or 0-many (zero or more values), then the attribute is not required for the object to pass schema checking.
Optional attributes are not used by SIMS, or are for informational purposes only.

`country` Object Class

The `country` object class is used to define the country node. This is also the root node of the OSI tree. The `country` object class is defined as follows:

```
( OID — 2.5.6.2
NAME 'country'
SUPERIOR 'top'
MUST (
    countryName
    )
MAY (
    description $ searchGuide
    )
)
```

TABLE 3-1 Required `country` Attributes

Attribute	Description
`countryName`	(cis, 1, {}) 2-character country code. Defines the root node of the OSI tree.

TABLE 3-2 Optional `country` Attributes

Attribute	Description
`description`	(cis, 0 – many, {}) Free form text. Description about the country node in the directory. Usually the full name of the country that matches the country code set in `countryName`. For example, if `countryName`=US, the description could be set to `description`=United States of America.
`searchGuide`	(cis, 0 – many, {}) Used by X.500 clients in constructing search filters. Present here due to the heritage of LDAP schema.

`organization` Object Class

The `organization` object class is used to create the second level nodes in the DIT. Used with `domainRelatedObject` object class. The object class is defined as follows.

```
( OID — 2.5.6.4
NAME 'organization'
SUPERIOR 'top'
MUST (
    organizationName
    )
MAY (
    businessCategory $ description $ destinationIndicator $
    facsimileTelephoneNumber $ internationaliSDNNumber $
localityName
    $ physicalDeliveryOfficeName $ postOfficeBox $ postalAddress
    $ postalCode $ preferredDeliveryMethod $ registeredAddress
    $ searchGuide $ seeAlso $ state $ streetAddress $
telephoneNumber
    $ teletexTerminalIdentifier $ telexNumber $ userPassword $
    x121Address
    )
)
```

TABLE 3-3 Required `organization` Attributes

Attribute	Description
`organizationName`	(cis, 0 - many, { }) Name of the organization associated with this group.

TABLE 3-4 Optional `organization` Attributes

Attribute	Description
`businessCategory`	(cis, 0 - many, {}) Business classification for the organization.
`description`	(cis, 0 – many, {}) Free form text. Description about the organization node in the directory. Usually the full name of the organization that is associated with the value of attribute `organizationName` for this entry. For example, if `organizationName`=Sun, the description can be set to `description`=Sun Microsystems, Inc.
`destinationIndicator`	(cis, 0 - many, {}) Country and city address information.
`facsimileTelephoneNumber`	(tel, 0 - many, {}) Fax telephone number of the distribution list.
`internationaliSDNNumber`	(tel, 0 - many, {}) International ISDN number of the distribution list.
`localityName`	(cis, 0 – many, {}) Locality name.
`physicalDeliveryOfficeName`	(cis, 0 - many, {}) Mail stop.
`postOfficeBox`	(cis, 0 - many, {}) Post office box.
`postalAddress`	(cis, 0 - many, {}) Postal address.
`postalCode`	(cis, 0 - many, {}) Postal code.
`preferredDeliveryMethod`	(cis, 0 - 1, {}) Preferred delivery method of communication.
`registeredAddress`	(cis, 0 - many, {}) Registered postal address.
`searchGuide`	(cis, 0 – many, {}) Used by X.500 clients to construct search filters. Present here due to the heritage of LDAP schema.
`seeAlso`	(dn, 0 - many, {}) Distinguished name of an entry to consult for further information.
`st`	(cis, 0 - many, {}) Full name of state or province (`stateOrProvinceName`).
`streetAddress`	(cis, 0 - many, {}) Street address associated with this organization.
`telephoneNumber`	(tel, 0 - many, {}) Telephone number in international format.
`teletexTermicalIdentifier`	(cis, 0 - many, {}) Teletex terminal ID and optional parameters for a teletex terminal. $ separated string.
`telexNumber`	(cis, 0 - many, {}) Telex number, country code, and answer back code for a teletex terminal.
`userPassword`	(protected / sunds, 0 - many, { }) Encrypted string representing the password of the organization node. In Sun Directory Server, the supported encryption scheme used is crypt or sunds. For CRAM-MD5 support the encryption scheme used in the SunDS 3.1 should be sunds.
`x121Address`	(cis, 0 - many, {}) An address as defined by the ITU recommendation X.121.

`organizationalUnit` Object Class

The `organizationalUnit` object class is used to create the container entries of the primary DIT. These entries are the organizational unit (ou) containers corresponding to an OSI tree based on geography (east, west, UK, Russia, and so on) or functional units (engineering, marketing, and so on). The ou entry is created using the `organizationalUnit` object class. Each organization unit is required to have three more ou entries — people, groups, and services. The object class is defined as follows.

```
( OID – 2.5.6.5
NAME 'organizationalUnit'
SUPERIOR 'top'
STRUCTURAL
MUST (
    organizationalUnitName
    )
MAY (
    businessCategory $ description $ destinationIndicator $
    facsimileTelephoneNumber $ internationaliSDNNumber $
localityName
    $ physicalDeliveryOfficeName $ postOfficeBox $ postalAddress $
    postalCode $ preferredDeliveryMethod $ registeredAddress $
    searchGuide $ seeAlso $ state $ streetAddress $ telephoneNumber
$
    teletexTerminalIdentifier $ telexNumber $ userPassword $
    x121Address
    )
)
```

TABLE 3-5 Required `organizationalUnit` Attributes

Attribute	Description
`organizationUnitName`	(cis, 0 - many, {admin,ms,mta,webaccess}) Name of the organization unit represented by this entry, for example, `ou=Engineering`.

TABLE 3-6 Optional `organizationalUnit` Attributes

Attribute	Description
`businessCategory`	(cis, 0 - many, {}) Business classification for the organization unit.
`description`	(cis, 0 – many, {}) Free form text. Description about the organization node in the directory. Usually the full name of the organization that associated with the value of attribute `organizationUnitName` for this entry. For example, if `organizationUnitName`=Engineering, the description could be set to `description`=All of engineering in Sun Microsystems, Inc.
`destinationIndicator`	(cis, 0 - many, {}) Country and city address information.
`facsimileTelephoneNumber`	(tel, 0 - many, {}) Fax telephone number of the distribution list.
`internationaliSDNNumber`	(tel, 0 - many, {}) International ISDN number of the distribution list.
`localityName`	(cis, 0 – many, {}) Locality name.
`physicalDeliveryOfficeName`	(cis, 0 - many, {}) Mail stop.
`postOfficeBox`	(cis, 0 - many, {}) Post office box.
`postalAddress`	(cis, 0 - many, {}) Postal address.
`postalCode`	(cis, 0 - many, {}) Postal code.
`preferredDeliveryMethod`	(cis, 0 - 1, {}) Preferred delivery method of communication.
`registeredAddress`	(cis, 0 - many, {}) Registered postal address.
`searchGuide`	(cis, 0 – many, {}) This attribute is for use by X.500 clients in constructing search filters. Present here due to the heritage of LDAP schema.
`seeAlso`	(dn, 0 - many, {}) Distinguished name of an entry to consult for further information.
`st`	(cis, 0 - many, {}) Full name of state or province (`stateOrProvinceName`).
`streetAddress`	(cis, 0 - many, {}) Street address associated with this organization.
`telephoneNumber`	(tel, 0 - many, {}) Telephone number in international format.
`teletexTermicalIdentifier`	(cis, 0 - many, {}) Teletex terminal ID and optional parameters for a teletex terminal. $ separated string.
`telexNumber`	(cis, 0 - many, {}) Telex number, country code, and answer back code for a teletex terminal.
`userPassword`	(protected / sunds, 0 - many, { }) Encrypted string representing the password of the organization node. In Sun Directory Server, the supported encryption scheme used is crypt or sunds. For CRAM-MD5 support the encryption scheme used in the SunDS 3.1 should be sunds.
`x121Address`	(cis, 0 - many, {}) An address as defined by the ITU recommendation X.121.

`domain` Object Class

The `domain` object class is used to create all the container entries (except for the root entry) in the Domain Component (DC) tree. These entries are the domain component representing DNS domains. The object class is defined as follows.

```
(OID — 0.9.2342.19200300.100.4.13
NAME 'domain'
SUPERIOR 'top'
STRUCTURAL
MUST (
    domainComponent
    )
MAY (
    associatedName $ businessCategory $ description $
    destinationIndicator $ facsimileTelephoneNumber $
    internationaliSDNNumber $ locality $ organizationName $
    physicalDeliveryOfficeName $ postOfficeBox $ postalAddress $
    postalCode $ preferredDeliveryMethod $ registeredAddress $
    searchGuide $ seeAlso $ state $ streetAddress $ telephoneNumber
$
    teletexTerminalIdentifier $ telexNumber $ userPassword $
    x121Address
    )
)
```

TABLE 3-7 Required `domain` Attributes

Attribute	**Description**
`domainComponent`	(cis, 1, {admin,ma,mta,webaccess,spm}) Name of the associated DNS domain for this DC node. DNS domain *eng.sun.com* would be represented in the DIT by the following chain of nodes: `dc=eng,dc=sun,dc=com,o=<root_suffix>`. And the DNS domain associated with a physical system, like *jurassic.eng.sun.com* would be represented in the DIT by the following chain of nodes: `dc=jurassic,dc=eng,dc=sun,dc=com,o=<root_suffix>`.

TABLE 3-8 Optional `domain` Attributes

Attribute	Description
`associatedName`	(dn, 0 - many, {admin,ma,mta,webaccess,spm}) Links the organizational X.500 (OSI) DIT and the DNS (Domain Component) tree. Used to link the DNS hierarchy to the OSI hierarchy. Where such entries don't exist or `inetTreeStyle` (defined in `inetDomain` object class) is set to DC, user, group, and service entries should be looked up in the DC tree.
`businessCategory`	(cis, 0 - many, {}) Business classification for the domain.
`description`	(cis, 0 – many, {}) Free form text. Description about the domain node in the directory. Usually the full name of the domain that associated with the value of `domainComponent` for this entry. For example, if `domainComponent=Eng`, the description may be set to `description`=Engineering domain in Sun Microsystems, Inc.
`destinationIndicator`	(cis, 0 - many, {}) Country and city address information.
`facsimileTelephoneNumber`	(tel, 0 - many, {}) Fax telephone number of the distribution list.
`internationaliSDNNumber`	(tel, 0 - many, {}) International ISDN number of the distribution list.
`localityName`	(cis, 0 – many, {}) Locality name.
`physicalDeliveryOfficeName`	(cis, 0 - many, {}) Mail stop.
`postOfficeBox`	(cis, 0 - many, {}) Post office box.
`postalAddress`	(cis, 0 - many, {}) Postal address.
`postalCode`	(cis, 0 - many, {}) Postal code.
`preferredDeliveryMethod`	(cis, 0 - 1, {}) Preferred delivery method of communication.
`registeredAddress`	(cis, 0 - many, {}) Registered postal address.
`searchGuide`	(cis, 0 – many, {}) Used by X.500 clients in constructing search filters. Present here due to the heritage of LDAP schema.
`seeAlso`	(dn, 0 - many, {}) Distinguished name of an entry to consult for further information.
`st`	(cis, 0 - many, {}) Full name of state or province (`stateOrProvinceName`).
`streetAddress`	(cis, 0 - many, {}) Street address associated with this organization.
`telephoneNumber`	(tel, 0 - many, {}) Telephone number in international format.
`teletexTermicalIdentifier`	cis, 0 - many, {}) Teletex terminal ID and optional parameters for a teletex terminal. $ separated string.

TABLE 3-8 Optional `domain` Attributes *(Continued)*

Attribute	Description
`TelexNumber`	(cis, 0 - many, {}) Telex number, country code, and answer back code for a teletex terminal.
`userPassword`	(protected / sunds, 0 - many, { }) Encrypted string representing the password of the organization node. In Sun Directory Server, the supported encryption scheme used is crypt or sunds. For CRAM-MD5 support the encryption scheme used in the SunDS 3.1 should be sunds.
`x121Address`	(cis, 0 - many, {}) An address as defined by the ITU recommendation X.121.

`inetDomain` Object Class

The `inetDomain` object class is used to create those entries in the DC tree that correspond to a DNS domain. This object class is overlayed on nodes created with `domain` object class. The DC entry is created by using `domain` object classes. The object class is defined as follows.

```
(OID — TBD
NAME 'inetDomain'
AUXILIARY
MUST (
dnsDomainName $ inetTreeStyle
)
MAY (
    owner $ inetAuthorizedServices $ inetLabeledURI $
    inetDomainStatus
    )
)
```

TABLE 3-9 Required `inetDomain` Attributes

Attribute	Description
`dnsDomainName`	(cis, 1, {admin,ma,mta,webaccess,spm}) DNS domain name associated with this node in the DIT.
`inetTreeStyle`	(cis, 1, {admin,ma,mta,webaccess,spm}) Type of tree that is associated with this DNS domain. Takes the following values: • **DC** All users, groups, and service entries are found in the domain component tree. • **OSI** All users, groups, and service entries are found in OSI tree. Linkage is provided by the value of attribute `associatedName`.

TABLE 3-9 Required `inetDomain` Attributes *(Continued)*

Attribute	Description
`owner`	(dn, 1-many, {admin, spm}) Distinguished name of the delegated administrator(s) for this domain.
`inetLabeledURI`	(ces, 0-1, {admin, spm,mta,msma}) The LDAP URI points to the naming context in the OSI tree where the users, groups, and other objects reside. This is useful for sites that need to deploy an OSI style tree. NOTE: Using a DC/OSI tree is discouraged and should be used only when necessary. Components check the `inetTreeStyle` attribute of the domain entry to determine if the users are in DC or OSI tree. If the attribute indicates that the users are to be found in the OSI tree, the value of `inetLabeledURI` is used to determine the naming context to search. The syntax for this URL is specified in the RFC1959 and the relevant ABNF is reproduced below:
`inetAuthorizedServices`	(dn, 0 - many, {ma, mta, admin}) DN of the authorized services for this domain. List of services that users in this domain are permitted. This attribute explicitly calls out the set of services that a user is authorized to use. NOTE: If this attribute is missing from a domains entry, users are allowed to use all services listed in the user entry, that is, when doing the intersection between the set of services called out in user entry and domain entry, the domain entries `inetAuthorizedServices` list is considered the universal set. Supported values for service name are: • **imaps** allows access to secure IMAP based message access. • **imap** allows access to IMAP based message access . • **pop3** allows access to POP based message access. • **pop3s** allows access to secure POP based message access. • **smtps** allows access to secure SMTP server for message submission. • **sunw_webaccess** allows access to the Web Access server. • **sunw_calendar** allows access to Sun Calendar server.
`inetDomainStatus`	(cis, 0-1, {mta, ma, admin}) If missing from a user entry, the semantics are the same as if the value is active. Supported values are: • **active** All accounts in the domain are active. Subscribers may use all services granted by `inetAuthorizedServices` for domain AND subscribers. • **inactive** All accounts in the domain are inactive. Subscribers may not use any services granted by `inetAuthorizedServices` for domain AND subscribers. Where possible, service requests for subscribers must return transient failures. • **deleted** All accounts in the domain are marked as deleted. The accounts may remain in this state within the directory pending purging of deleted domain and all entries contained within. Service requests for subscribers in this domain must return permanent failures.

- inetLabeledURI

```
<ldapurl> ::= “ldap://” [<hostport>] “/” <dn> [“?” <attributes> [“?”
<scope> “?” <filter>

<hostport> ::= <hostname> [“:” <portnumber>]

<dn> ::= distinguished name (string) as defined in RFC 1779

<attributes> ::= NULL | <attributelist>
```

```
<attributelist> ::= <attributetype> | <attributetype> [“,”
<attributelist>
<attributetype> ::= a string as defined in RFC1777
<scope> ::= “base” | “one” | “sub”
<filter> ::= a string as defined in RFC1558
```

simsDomain Object Class

The simsDomain object class is used to create entries in the DC tree that correspond to a DNS domain. The DC entry is created by using domain, inetDomain, and simsDomain object classes. The object class is defined as follows.

```
( OID — TBD
NAME 'simsDomain'
AUXILIARY
MUST (
    simsDomainVersion
    )
MAY (
    rfc822Postmaster $ mailHosts $ preferredMailHost $
    domainDiskQuota $ mailQuota $ maxMailboxes $ maxEntries $
    maxDistributionLists $ simsRecursive
    )
)
```

TABLE 3-10 Required simsDomain Attributes

Attribute	Description
rfc822Postmaster	(cis, 0-many, {mta, admin}) rfc822 address of the postmaster.
mailHosts	(cis, 0-many, {admin,mta,spm}) List of fully qualified host names of mail servers that have routing responsibility for this domain. Used by MTA to build local routing tables. When a domain node has mailserver's name listed in mailhosts, it implies that all sub-domains are included in the routing scope of that mailserver. The alias database on the mailserver will be populated with users from all contained domains. NOTE: It is a provisioning error to have a mailserver listed in a domain nodes list of mailhosts when a superior domain node already has that mailserver in its list of mailhosts.
preferredMailHost	(cis, 0-1, {admin,spm}) Fully qualified host name of the preferred mail server for this hosted domain. When the delegated administrator adds a new user/distribution list, the new user/distribution list is assigned this value for their mailhost.
domainDiskQuota	(cis, 0-1, {ms, admin}) Disk quota in megabytes for this domain. Disk usage for all users in this hosted domain should not exceed this value.
maxMailBoxes	(cis, 0-1, {ma, admin}) Number of allowable mailboxes for this domain.
maxEntries	(cis, 0-1, {admin}) Number of allowable directory entries for this domain.
maxDistributionLists	(cis, 0-1, {mta, admin}) Number of allowable distribution lists for this domain.
simsRecursive	(cis, 0 – 1, {mta,admin,ms,spm}) Legal values for this attribute are 0 and 1. This attribute controls the bounds of the namespace. When set to 1 for any domain, it implies that users in the domain and all contained domains are in a flat namespace and all components must treat all users in that sub-tree as part of this domain. A value of 0 implies that only users in that domain are in the scope of the domains namespace and all sub-domains are separate namespaces whose bounds are determined by the simsRecursive flag of that domain. All domain provisioning tools must disallow creation of a domain node <sub>.<domain>.COM when the parent node <domain>.COM has simsRecursive=1. This would lead to overlapped namespace (parent domain says all users are within my scope and the child domain node says that all users beneath the child domain are in the child domains scope, hence leading to two namespaces claiming rights over the same users). If there is a compelling reason to create a sub-domain beneath a domain where simsRecursive=1, then the value of simsRecursive for the sub-domain must be set to 1. Additionally, a domain node marked with simsHosting=1 may not have simsRecursive=1.

Internet Mail User Object Classes

A user, including a SIMS email user, is represented by an entry in the directory. A user entry is extensible (as are all other directory entries) and may contain additional object classes/attributes once such schema extensions have been made in the directory. Take care to ensure that semantics of existing object classes are not changed by a schema extension.

An entry that stores user information for an email user consists of attributes drawn from — at a minimum — the following directory object classes.

The keyword in parentheses following the name of the object class, indicates whether the object class is standard, shared (by various services that use the directory data), or service-specific. If it is service-specific, the keyword is followed by the name of the service.

- `top` (standard) – Attributes for describing the classifications of a directory object. This is a structural object class and all other object classes inherit from this base class.
- `person` (standard) – Attributes for describing a person. Inherits from top.
- `organizationalPerson` (standard) – Attributes for describing a person belonging to an organization. Inherits from person.
- `inetOrgPerson` (proposed standard) – Same as `organizationalPerson` and also one that interacts with the Internet. Inherits from `organizationalPerson`.
- `inetSubscriber` (shared; auxiliary) – Attributes for describing a basic Sun Internet Services user. This is an auxiliary object class. All users who are provisioned for email, Web, ftp, and so on, are described using this object class and a combination of one or more service-specific auxiliary object classes. The `inetSubscriber` object class is an auxiliary class shared by several Sun products. It requires an `inetOrgPerson` structural object class, because a number of auxiliary object classes depend on attributes from `inetOrgPerson`.

  The `inetSubscriber` object class provides information needed to manage a subscriber of one or more Internet services (for example, sending email, retrieving received email, calendar access, and so on). This results in a single shared object that can be checked to determine which services a specific user is authorized to use. (Although it is beyond the scope of this chapter, the `inetSubscriber` object class is being used to support access to Internet services beyond the email domain (for example, http, news, and so on).
- `IinetMailUser` (service specific; SIMS; auxiliary) – Attributes for describing an email user. This is an auxiliary object class and is required for defining an email user. The `inetMailUser` object class, in conjunction with the auxiliary object classes `inetSubscriber`, `inetMailRouting,` and the structural object class `inetOrgPerson`, will be present in the LDAP directory entries for all users who will receive, send, or read Internet email. Internet email clients and servers should

use this object class to store and retrieve information related to storage of incoming email and sending of outbound email. All email users must have this object class.

- `imCalendarUser` (service specific; SICS, auxiliary) – Attributes for describing a calendar subscriber. This is an auxiliary class and is required for defining users of Sun Internet Calendar Server.
- `inetAdministrator` (shared; auxiliary) – Attributes for describing an administrator for SIMS.
- `inetMailRouting` (service specific; SIMS; auxiliary) – Attributes containing the required routing information common to all Internet email recipients.
- `inetMailGroup` (service specific; SIMS; auxiliary) – Attributes that are key to determining how the mail is processed by the MTAs. Additionally, the `inetMailRouting` object class determines how messages are routed through the mail system.

A detailed explanation of the attributes in this object class (including the valid range of values for the attribute, the effect on the behavior of SIMS as a result of changing the value, and the syntax for the attribute), follows the definition of the object class. The attribute list can have:

- Required attributes that are used either for the basic functionality of SIMS (that is, required for the components to function) or by one or more service when extended features of these products are used.
- Reserved attributes reserved for future use by SIMS and should not be used by the end user for other purposes.
- Optional attributes that are not used nor planned to be used by SIMS.

Attribute names are followed by — within parenthesis — attribute syntax and list of services that depend on the attribute.

Inherited Object Classes and Attributes

The following attributes are used in this specification but are defined in other specifications: `uid`, `userPassword`, `givenName`, `commonName`, and `surname`.

- `uid` (cis, 1, {client, msma, admin}) attribute – The name, unique within a domain, used by a subscriber to log in to a computer system. The `uid` attribute must be used to authenticate to the email message access service before the user may read messages in their mailbox(es).
- `userPassword` (cis, 1, {client, msma, admin}) attribute – The password used by the subscriber to authenticate to a server for access to a particular service.
- `givenName` (ces, 0-many, {admin}) attribute – Used to hold the part of a person's name which is not their surname or middle name.

- surname (ces, 0-1, {admin}) attribute – Used to hold a person's last, or family, name.
- commonName (ces, 0-many, {admin}) attribute – Used to hold the concatenation of a person's first and last (or family) name. In the directory the commonName of a person is a naming attribute. Because names are not always unique, provisioning software may optionally transform a non-unique commonName into a name that is unique within a domain; it may do this by further concatenating the value of the uid attribute to the default commonName value, and then using that now-unique value as the naming attribute. This is acceptable because the commonName attribute is defined as multi-valued.

top Object Class

The top object class is the base class for all other structural object classes used by Sun Internet Services. The object class is defined as follows.

```
(OID — 2.5.6.0
NAME 'top'
STRUCTURAL
MUST (
    objectClass
    )
)
```

TABLE 3-11 describes the required attributes for the top object class.

TABLE 3-11 Required top Attributes

Attribute	Description
objectClass	(cis, 1 – many, {mta, ma, smcs, admin}) The object classes used to define a directory entry. For every node in the directory (vertex and leaf nodes), we have to use one structural object class and zero to many auxiliary object classes.

person Object Class

```
(OID — 2.5.6.6
NAME 'person'
STRUCTURAL
SUPERIOR 'top'
MUST (
    surname $ commonname
    )
MAY (
    description $ seeAlso $ telephoneNumber $ userPassword
    )
)
```

TABLE 3-12 Required `person` Attributes

Attribute	Description
`commonname`	(cis, 1 - many, {mta, admin}) Users full name. There can be more than one cn attribute for a user although each attribute is required to be unique within a user record.
`surname`	(cis, 1 – many, {}) Users last or family name.

The following are the optional person attributes.

TABLE 3-13 Optional `person` Attributes

Attribute	Description
`description`	(cis, 0 –many, {}) Free form text. Description of user entry.
`seeAlso`	(dn, 0 - many, {}) Distinguished name of an entry to consult for further information.
`telephoneNumber`	(tel, 0 - many, {}) Telephone number in international format.
`userPassword`	(protected / sunds, 0 - many, {admin, imta, ms, ftp, calendar}) Encrypted string representing the users password. In Sun Directory Server, the supported encryption scheme used is crypt or sunds. For CRAM-MD5 support the encryption scheme used in the SunDS 3.1 should be sunds.

organizationalPerson Object Class

Attributes for describing a person belonging to an organization.)

```
(OID — 2.5.6.7
NAME 'organizationalPerson'
STRUCTURAL
SUPERIOR 'person'
MAY (
    destinationIndicator $ facsimileTelephoneNumber $
    internationaliSDNNumber $ localityName $
organizationalUnitName
    $ physicalDeliveryOfficeName $ postOfficeBox $ postalCode $
    preferredDeliveryMethod $ registeredAddress $ st $ street $
    telephoneNumber $ telexTerminalIdentifier $ title $
x121Address
    ))
```

TABLE 3-14 Optional organizationalperson Attributes

Attribute	Description
destinationIndicator	(cis, 0 - many, {}) Country and city address information.
facsimileTelephoneNumber	(tel, 0 - many, {}) Fax telephone number of the distribution list.
internationaliSDNNumber	(tel, 0 - many, {}) International ISDN number of the distribution list.
localityName	(cis, 0 – many, {}) Locality name.
organizationalUnitName	(cis, 0 – 1, {}) Organizational Unit name.
physicalDeliveryOfficeName	(cis, 0 - many, {}) Mail stop.
postOfficeBox	(cis, 0 - many, {}) Post office box.
postalAddress	(cis, 0 - many, {}) Postal address.
postalCode	(cis, 0 - many, {}) Postal code.
preferredDeliveryMethod	(cis, 0 - 1, {}) Preferred delivery method of communication.
registeredAddress	(cis, 0 - many, {}) Registered postal address.
st	(cis, 0 - many, {}) Full name of state or province (stateOrProvinceName).
street	(cis, 0 - many, {}) Street name.
teletexTermicalIdentifier	(cis, 0 - many, {}) Teletex terminal ID and optional parameters for a teletex terminal. $ separated string.

TABLE 3-14 Optional `organizationalperson` Attributes *(Continued)*

Attribute	Description
`telexNumber`	(cis, 0 - many, {}) Telex number, country code and answer back code for a teletex terminal.
`title`	(cis, 0 - many, {}) Contains the title of a person in their organizational context.
`x121Address`	(cis, 0 - many, {}) An address as defined by the ITU recommendation X.121.

`inetOrgPerson` Object Class

Attributes use to describe a person using Internet services.

```
(OID – 2.16.840.1.113730.3.2.2
NAME 'inetOrgPerson'
STRUCTURAL
SUPERIOR 'organizationalPerson'
MAY (
   audio $ businessCategory $ carLicense $ departmentNumber
   $ employeeNumber $ employeeType $ givenName $ homePhone
   $ homePostalAddress $ initials $ jpegPhoto $ labeledURI $ mail
   $ manager $ mobile $ pager $ photo $ roomNumber $ secretary $ uid
   $ userCertificate $ x500uniqueueIdentifier $ preferredLanguage
   $ userSMIMECertificate
   )
)
```

TABLE 3-15 Optional `inetOrgPerson` Attributes

Attribute	Description
`audio`	(bin, 0 - many, {}) – Audio clip.
`businessCategory`	(cis, 0 - many, {}) – Business classification for the user.
`carLicense`	(cis, 0 - many, {}) – Vehicle license plate number.
`departmentNumber`	(cis, 0 - many, {}) – Numeric or alpha-numeric code for department to which a person belongs.
`employeeNumber`	(cis, 0 - 1, {}) – Numeric or alpha-numeric identier assigned to a person, typically based on order of hire or association with an organization. Single valued.
`employeeType`	(cis, 0 - many, {}) – Used to identify employer to employee relationship. Typical values used will be contractor, employee, intern, temp, external, and unknown, but any value may be used.

TABLE 3-15 Optional `inetOrgPerson` Attributes *(Continued)*

Attribute	Description
`givenName`	(cis, 0 - many, {}) – First name of the user.
`homePhone`	(tel, 0 - many, {}) – Home telephone number.
`homePostalAddress`	(cis, 0 - many, {}) – Home postal address.
`initials`	(cis, 0 - many, {}) – Initials attributes contains the initials of some or all of an individuals names, but not the surname(s).
`jpegPhoto`	(bin, 0 - many, {}) – Photograph stored in JPEG format.
`labeledURI`	(ces, 0 - many, {}) – Uniform resource identifier.
`mail`	(cis, 0-many, {mta, admin}) – The user's advertised email address (RFC 822 format). Also know as `preferredRfc822Originator`.
`manager`	(dn, 0-1, { }) – Distinguished name of manger.
`mobile`	(tel, 0 - many, {}) – Mobile telephone number.
`pager`	(tel, 0 - many, {}) – Pager number.
`photo`	(bin, 0 - many, {}) – Photograph associated with this user.
`roomNumber`	(cis, 0 - many, {}) – Room number.
`secretary`	(dn, 0-1, { }) – Distinguished name of secretary.
`uid`	(cis, 0 – 1, {mta, ma, admin, calendar}) – The login identifier of the user. The naming context within which this is required to be unique is the naming context associated with containing DNS domain.
`userCertificate`	(cis, 0 – many, { }) – User certificate.
`x500UniqueIdentifier`	(cis, 0 – many, { }) – `x500UniqueueIdentifier` attribute is used to distinguish between objects when a distinguished name has been reused.
`uid`	(cis, 0 – 1, {mta, ma, admin, calendar}) – The login identifier of the user. The naming context within which this is required to be unique is the naming context associated with containing DNS domain.
`userCertificate`	(cis, 0 – many, { }) – User certificate.

TABLE 3-15 Optional `inetOrgPerson` Attributes *(Continued)*

Attribute	Description
`x500UniqueIdentifier`	(cis, 0 – many, { }) – Used to distinguish between objects when a distinguished name has been reused.
`preferredLanguage`	(cis, 0 – 1, { }) – Used to indicate an individual's preferred written or spoken language. This is useful for international correspondence or human-computer intercation. Values must conform to the definition of the Accept-Language header field defined in RFC2068, with one exception: the sequence "Accept-Language" ":" should be omitted. This is a single-valued attribute.
`userSMIMECertificate`	(bin, 0 - many, {}) – S/MIME signed message with a zero length body. This attribute is stored and requested in binary form. It contains the person's entire certificate chain and the signed attribute that describes their algorithm capabilities, stored as an `octetString`. If available, this attribute is preferred over the `userCertificate` attribute for S/MIME applications.

`inetSubscriber` *Object Class*

The `inetSubscriber` object class is an auxiliary object class used to define an Internet subscriber. SIMS uses this object class along with `inetMailRouting` and `inetMailUser` to define an email user. The is object class defined as follows:

```
( 1.3.6.1.4.1.42.2.27.3.2.1
NAME 'inetSubscriber'
SUP top
AUXILIARY
MUST (
    uid
    )
MAY (
    inetAuthorizedServices $ inetSubscriberHttpURL $
    inetSubscriberStatus
    )
)
```

TABLE 3-16 Optional `inetSubscriber` Attributes

Attribute	Description
`inetAuthorizedServices`	(cis, 0-many, {client, mta, msma, admin}) A list of tokens representing services that this user is authorized to access. If this attribute is missing from a user entry, then the user has permission to use all supported Internet services. If more granular authorization is wanted, provisioning tools should add the tokens representing services available to the user. It is recommended that a directory access control rule be added to the system to restrict the user's ability to modify this attribute. The tokens defined by this document are: • **imaps** Access to secure IMAP based message access. • **imap** Access to IMAP based message access. • **pop3** Access to POP based message access. • **pop3s** Access to secure POP based message access • **smtps** Access to secure SMTP server for message submission. • **sunw_webaccess** Access to the Web Access server. • **sunw_calendar** Access to Sun Calendar server.
`inetSubscriberHttpURL`	(ces, 0-many, {}) Contains HTTP-based URL's for the subscribers web page(s).
`inetSubscriberStatus`	(cis, 0-1, {client, mta, msma, admin}) Specifies the status of a subscribers account with regard to global access. Allows the Internet Service Provider to temporarily suspend and re-enable access, or to permanently remove access, by the subscriber to the account. This attribute takes one of three values. If this attribute is missing from a user entry, the semantics are the same as if the value is `active`. Supported values are: • **active** Account is active. The subscriber may use all accesses granted by `inetAuthorizedServices`. • **inactive** Account is inactive. The subscriber may not use any services granted by `inetAuthorizedServices`. Service requests for a user marked as inactive must return transient failures. • **deleted** Account is marked as deleted. The account may remain in this state within the directory pending purging of deleted users. Service requests for a user marked as deleted must return permanent failures. Users marked inactive can be made active, that is, service can be restored by changing the value of this attribute to `active`. Users marked deleted, may require further actions outside the context of the Directory to re-instate services. For example, if their mailboxes have been archived to tape, or even removed, they might not be available immediately (if at all) if the account is made active.

`inetMailUser` *Object Class*

This auxiliary object class is used to overlay LDAP entries defined by `inetOrgPerson`, and it allows that user to receive and send email. Two other auxiliary object classes are used along with `inetMailUser` for a user to become a SIMS user: `inetMailRouting` and `inetSubscriber`.

The `inetMailUser` object class is defined as follows:

```
( 1.3.6.1.4.1.42.2.27.2.2.3
NAME 'inetMailUser'
SUP top
AUXILIARY
MUST (
    inetMailUserVersion
    )
MAY (
    datasource $ mailAutoReplyStartDate $
    mailAutoReplyExpirationDate $ mailAutoReplyTimeout $
    mailAutoReplySubject $ mailAutoReplyText $
    mailAutoReplyTextInternal $ mailDeliveryFile $
    mailDeliveryOption $ mailFolderMap $
    mailForwardingAddress $ mailMessageStore $
    mailProgramDeliveryInfo $ mailQuota $ userDefinedAttribute1 $
    userDefinedAttribute2 $ userDefinedAttribute3 $
    userDefinedAttribute4
    )
)
```

TABLE 3-17 Optional `inetMailUser`: Membership Attributes

Attribute	Description
`datasource attribute`	(cis, 0-1, {admin}) Free form text that describes the source or identifier of the provisioning tool.
`inetMailUserVersion`	(ces, 1, {admin}) The version tag of this object class. This attribute exists so that LDAP clients supporting Internet email services may retrieve LDAP objects that support a particular revision of schema that they want to support. The starting value of version tags is “1.0”, and any change to this object class in the future must increment the `inetMailUserVersion` attribute value.
`mailAutoReplyStartDate`	(generalizedTime, 0-1, {client, mta} Specifies when an MTA should enable automatic replies to incoming email for a user with this attribute set.

TABLE 3-17 Optional `inetMailUser`: Membership Attributes *(Continued)*

Attribute	Description
`mailAutoReplyExpirationDate`	(generalizedTime, 0-1, {client, mta}) Specifies the date on which to disable automatic replies to incoming email for a user with this attribute set.
`mailAutoReplyTimeout`	(cis, 0-1, {client, mta}) For a user with `mailDeliveryOption` set to `autorepl`, contains the duration, in hours, between successive auto-replies to incoming email from a specific sender. An implementation may choose to treat aliases for the same recipient as distinct (separate) senders. The MTA must not send auto-replies to distribution lists.
`mailAutoReplySubject`	(cis, 0-1, {client, mta} The subject line of an auto-reply message. If it contains $SUBJECT then the token is replaced by the subject line of the incoming message.
`mailAutoReplyText`	(cis, 0-1, {client, mta} The body of the auto-reply message. If it contains the tokens $SUBJECT or $BODY, then these are replaced by the subject or the body of the inbound message. Use '$' as a line separator.
`mailAutoReplyTextInternal`	(cis, 0-1, {client, mta}) The body of the auto-reply message for internal auto-replies. Only those senders within the same domain receive the `mailAutoReplyTextInternal`. If this attribute contains the tokens $SUBJECT or $BODY, then these are replaced by the subject or the body of the inbound message. Use '$' as a line separator.
`mailDeliveryFile`	(ces, 1-many, {mta}) The fully qualified path name of a file to which incoming messages are appended. This file must be accessible for writing from the file system on the user's mail host.

TABLE 3-17 Optional `inetMailUser`: Membership Attributes *(Continued)*

Attribute	Description
`mailDeliveryOption`	(cis, 1-many, {mta}) Specifies one or more delivery options for inbound email to a designated recipient. While inbound messages can be delivered into multiple message stores, message access servers can read messages from only one of them (the mail store from which messages are read is specified using the `mailFolderMap` attribute). The Message Transfer Agent uses this attribute to determine the targets of message delivery for all messages submitted to this individual recipient. The attribute is also used by the `inetMailGroup` object class. The value of this attribute can take one of a specified set of options; the subset valid for individual recipients are described as follows: • `mailbox` - Deliver mail to a vendor specific / high performance Message Store mailbox. The `mailFolderMap` attribute specifies the mail store from which a Message Access agent would retrieve delivered mail. For example, in the unbundled Sun Internet email product, provisioning a user to read messages from the Sun Message Store would require setting the `mailDeliveryOption` to `mailbox`, and the associated `mailFolderMap` attribute to `Sun-MS`. • `shared` - Applies only to the `inetMailGroup` object class. • `native` - Deliver mail to a local sendmail-style file system mailbox (also known as the */var/mail box*). If `mailDeliveryOption` is set to `native`, then the `mailFolderMap` attribute must be set to `UNIX V7` for the user to read messages from the native message store using the Sun Internet email product's message access services. • `autoreply` - Deliver mail to an auto-reply facility. When this value is set the behavior of the autoreply feature of the MTA will be controlled by the following `inetMailUser` attributes: `mailAutoReplyStartDate`, `mailAutoReplyExpirationDate`, `mailAutoReplyTimeout`, `mailAutoReplySubject`, `mailAutoReplyText`, and `mailAutoReplyTextInternal`. • `program` - Deliver mail to another program. • `forward` - Forward incoming mail to another RFC-822 compliant address. Refer to the `mailForwardingAddress` for related information. MTAs must be able to parse options other than those above, although a particular MTA may not be able to support such options. This is so that vendors can use attribute values other than those specified above. In such cases, it is recommended that the name be prefixed with a vendor-specific name, such as a stock symbol. • `hold` - By setting `hold`, all other mail delivery options are overridden. All incoming messages for this user are sent to the hold channel.

TABLE 3-17 Optional `inetMailUser`: Membership Attributes *(Continued)*

Attribute	Description
`mailFolderMap`	(cis, 0-1, {msma, admin}) The message store for a user's mail folders. Message access servers (imap server, pop server, and so on) use this attribute to determine a user's primary mailbox. An MTA may deliver a message into multiple locations and message access servers have to be told the default mailbox of the user. Supported values in the unbundled Sun Internet email product are: • UNIX V7 - sendmail-style mail store. Also known as the Berkeley style /var/mail message store. • Sun-MS - Sun Message Store. A high performance message store accessed via POP or IMAP protocols.

`mailForwardingAddress`	(cis, 0-many, {mta, admin}) Specifies that the MTA should forward email to the specified Internet email address (RFC822 format). For the MTA to forward the email to these addresses, the `mailDeliveryOption` attribute should include the value `forward` in addition to any other delivery options.
	For example, if a user wants to forward mail to another address, then the directory entry for the user has the first block of values for `mailForwardingAddress` and `mailDeliveryOption`. However if the user wants to continue receiving mail on their default server and forward a copy of every message to another address then the directory entry would have the second block of values. Example:
	`mailDeliveryOption: forward`
	`mailFolderMap: Sun-MS`
	`mailForwardingAddress: <RFC-822 address>`
	
	`mailDeliveryOption: forward`
	`mailDeliveryOption: mailbox`
	`mailFolderMap: Sun-MS`
	`mailForwardingAddress: <RFC-822 address>`

`mailMessageStore`	(ces, 0-1, {mta, admin}) The file system location for a user's INBOX. This applies only when a `mailDeliveryOption` is set to native. The MTA will deliver incoming messages to this file. The file system location is in the context of the mail host. If this value is missing and the user's `mailDeliveryOption` is set to native, then a default of /var/mail is used by the server. This attribute specifies only the name of the directory; to derive the full name of the INBOX, the value of the `uid` attribute is appended to the directory name.
`mailProgramDeliveryInfo`	(ces, 0-many, {mta, admin}) Specifies one or more named commands to use in email delivery. The valid named commands must be defined by the MTA for secure operation. These named commands are defined by system administrators of the mail server and are mapped to an executable (with zero or more options) which processes the messages addressed to the user. These programs are installed on the mail server and the MTA must check that the program listed in the user's entry is on the approved list before it starts the program.

TABLE 3-17 Optional `inetMailUser`: Membership Attributes *(Continued)*

Attribute	Description
`mailQuota`	(cis, 0-1, {mta, msma, admin}) Specifies the maximum size (in bytes) of a user's message store. Note that this includes the Inbox and all other mailboxes or folders which the user may have in the message store. A value of minus one (-1) or a missing value denotes no limit on the cumulative size of messages in a user's INBOX and/or folder collection. A value of minus two (-2) implies that the system or domain default is used. The default unit of bytes may be overridden by using one of the tags listed below prefixed by the size: • <size>K - size is specified in kilo bytes • <size>M - size is specified in mega bytes • <size>G - size is specified in giga bytes • <size>T - size is specified in tera bytes
`userDefinedAttribute1`	(cis, 0-many, {}) May be used by the user.
`userDefinedAttribute2`	(cis, 0-many, {}) May be used by the user.
`userDefinedAttribute3`	(cis, 0-many, {}) May be used by the user.
`userDefinedAttribute4`	(cis, 0-many, {}) May be used by the user.

`inetAdministrator` Object Class

The `inetAdministrator` object class is used to tag `inetSubscribers` who have administrative capabilities. The object class name itself serves as the tag, and the `inetAdministeredServices` attribute is used to identify the type of service and the management scope for when administrative privileges are granted.

```
OID – TBD
Name  ‘inetAdministrator’
SUP top
AUXILIARY
MAY (
    inetAdministeredServices
    )
```

TABLE 3-18 Optional `inetAdministrator` Attributes

Attribute	Description
`inetAdministeredServices`	(cis, 0-many, {sia,ma,spm,admin}) A multi-valued attribute enumerating the services that a particular administrator is authorized to administer. This attribute is checked by a component before granting administrative access to that component.

`imCalendarUser` Object Class

Use the `imCalendarUser` object class to define the calendar server attributes for an `inetSubscriber`. Use in conjunction with `inetorgperson` and `inetSubscriber`.

```
(OID — TBD
NAME 'imCalendarUser'
AUXILIARY
MUST (
    uid $ userPassword $ imCalendarHost $ imCalendarUserVersion
    )
MAY (
    imCalendarName
    )
)
```

This object class promotes `uid` (alias `userid`) and `userPassword` to required attributes. These attributes are previously defined in `inetOrgPerson` and `person` object classes respectively.

TABLE 3-19 Required `imCalendarUser` Attributes

Attribute	Description
`uid`	(cis, 0 – 1, {mta, ma, admin, calendar}) The login identifier of the user. The naming context within which this is required to be unique is the naming context associated with containing DNS domain.
`userPassword`	(protected / sunds, 0 - many, {admin, imta, ms, ftp, calendar}) Encrypted string representing the users password. In Sun Directory Server, the supported encryption scheme used is crypt or sunds. For CRAM-MD5 support the encryption scheme used in the SunDS 3.1 should be sunds.
`imCalendarHost`	(cis, 0 - 1, {admin, calendar}) Fully qualified host name of the calendar server. This calendar server provides access to the users calendar.
`imCalendarUserVersion`	(ces, 0 – 1, {admin}) Version tag of this object class. This is a new attribute added to the object classes. The starting value of version tags is 2.0 and provisioning tools should set this attributes value to 2.0.

TABLE 3-20 Optional `imCalendarUser` Attributes

Attribute	Description
`imCalendarName`	(cis, 0 – 1, {admin, calendar}) Name of the calendar object associated with the subscriber. This calendar object resides on the calendar server called out in `imCalendarHost` attribute.

Internet Mail Distribution List Object Classes

Distribution lists are of groups of users and groups to which message can be sent. An email distribution list is represented by an entry in the directory. An entry, which stores distribution list information, consists of attributes drawn from these object classes:

- `groupOfUniqueNames` – Attributes for describing a collection of user objects. Inherits from `top` and is a structural object class. All SIMS email distribution lists are provisioned using this object class and the auxiliary object classes `inetMailRouting` and `inetMailGroup`.
- `inetMailGroup` – Attributes for describing an email distribution list. All distribution lists are provisioned using auxiliary object classes and is required for defining a SIMS distribution list.

A distribution lists entry is extensible and may contain attributes from additional object classes once such object classes have been defined in the directory schema.

URL for Attributes Containing Addresses

Several `inetMailGroup` attributes contain either RFC-822 style mail addresses or distinguished names (DN) of LDAP entries. This is permitted because `inetMailGroup` is both an LDAP and email entity, and it is appropriate to allow both types of addresses. The attributes `errorsTo`, `moderator`, `authorizedSubmitter`, `unauthorizedSubmitter`, use URL's [URL] to allow this dual use. When preceded by *ldap:///* the entry is used as an LDAP entry with the remaining value treated as the distinguished name of the entry. When preceded by *mailto:* the entry is interpreted as an RFC-822 address.

A missing prefix of *ldap:///* or *mailto:* for the entry is assumed to be an RFC-822 address.

The URL has the form:

ldap:///[<server>[:<port>]]/<baseDN>?[<attrs>]?<scope>?<filter>

- *attrs* is not applicable for this use and is ignored.
- Default value for *server:port* is the LDAP server being used by the MTA.
- The *baseDN* specifies the base for the search; if not present, the default is the *baseDN* used by the MTA.
- *scope* defines levels of the directory tree to be searched relative to the specified search base; its default value is *base*.
- The default for *filter* is *(mail=*)*, because you want to include only entities in the distribution list that can accept mail.

`groupOfUniqueNames` Object Class

The `groupOfUniqueNames` object class contains attributes for describing a collection of directory entries (namely users and other groups). This object class inherits from top and is a structural object class. This structural class is used along with `inetMailGroup` and `inetMailRouting` to provision Sun Internet Mail Server distribution lists.

```
(OID — 2.5.6.17
NAME 'groupOfUniqueNames'
SUPERIOR 'top'
MUST (
    commonname $ uniqueMember
    )
MAY (
    businessCategory $ description $ organizationName $
    organizationalUnitName $ owner $ seeAlso
    )
)
```

TABLE 3-21 Required `groupOfUniqueNames` Attributes

Attribute	Description
`commonname`	(cis, 1 - many, {mta, admin}) A distribution lists common name. This names is used for display only.
`uniqueMember`	(dn, 0 - many, {mta, ma, admin}) Distinguished names of members of this list. These users have to be defined in the directory for them to receive email messages sent to the list.
`owner`	(dn, 0 - many, {admin,spm}) Distinguished name of the owner(s) of this group. Owners have the rights to modify the group membership.

TABLE 3-22 Optional `groupOfUniqueNames` Attributes

Attribute	Description
`businessCategory`	(cis, 0 - many, {}) Business classification for the group.
`description`	(cis, 0 - many, { }) Description of the group.

TABLE 3-22 Optional `groupOfUniqueNames` Attributes

Attribute	Description
`organizationName`	(cis, 0 - many, { }) Name of the organization associated with this group.
`organizationUnitName`	(cis, 0 - many, { }) Name of the organizational unit associated with this group.
`seeAlso`	(dn, 0 - many, {}) Distinguished name of an entry to consult for further information about the group.

`inetMailGroup` Object Class

The `inetMailGroup` object class contains attributes useful for describing an email distribution list. SIMS requires this object class for defining a distribution list. All distribution lists are provisioned using this auxiliary object class, the `inetMailRouting` auxiliary object class, and the structural object class `groupOfUniqueNames`. These object classes are overlayed on entries created with the `groupofUniqueNames` object class. This object class is defined as follows.

```
( 1.3.6.1.4.1.42.2.27.2.2.2
NAME 'inetMailGroup'
SUP top
AUXILIARY
MUST (
    inetMailGroupVersion
    )
MAY (
    errorsTo $ joinable $  moderator $ multiLineDescription $
    requestsTo $ seeAlso $ suppressEmailError $ userPassword $
    authorizedDomain $ authorizedSubmitter $ dataSource $
    inetGroupStatus $ expandable $ mailDeliveryFile $
    mailDeliveryOption $ mailProgramDeliveryInfo $
rfc822Mailmember $
    unauthorizedDomain $ unauthorizedSubmitter $ membershipFilter
    )
)
```

The `inetMailGroup` object class attributes are grouped into the following categories.

Mail Processing Attributes

Several `inetMailGroup` attributes are key to determining how the mail is processed by the MTAs. Additionally, `inetMailRouting` determines how messages are routed through the mail system. One attribute indicates the version of the object class itself.

TABLE 3-23 Required `inetMailGroup` Attributes

Attribute	Description
`inetMailGroupVersion`	(ces, 0-1, {admin}) Version tag of this object class. This attribute must be set when an entry is created using this object class. The starting (current) value of version tag is 1.0.

TABLE 3-24 Optional `inetMailGroup` Attributes

Attribute	Description
`errorsTo`	(ces, 0-1, {admin, MTA}) Indicates the address to which list errors are sent. When a list is expanded, the original return address in the envelope is replaced by this address. The intent is for lists errors to be sent to the owner of the list, rather than the message originator, who generally has no control over the contents of the list. The Requirements for Internet Hosts [RFC1123] specify that all MTAs should support a mechanism where a list is expanded, but with the original return address preserved. This is referred to by the RFC as "aliasing." This can be achieved by omitting the `errorsTo` attribute. This is different from the `rfc822MailAlias` attribute, which is an alternative name for a single user or list, and does not cause any kind of address list expansion.
`requestsTo`	(ces, 0-many, {mta, admin}) Distribution list addresses are specified using the `mail` and `rfc822MailAlias` attributes of the `inetMailRouting` object class. Addresses of this form may be represented as <addr_local_part>@<domain_part>. Messages sent to an address constructed by adding "-request" to the <addr_local_part> of the distribution list address will be delivered (forwarded) to the address(es) specified in the `requestsTo` attribute. For example, a distribution list with the following addresses: mail: *football@sun.com* rfc822MailAlias: *football-fans@sun.com* requestsTo: *mailto:john.doe@isp.net* would forward messages addressed to *football-request@sun.com* and *football-fans-request@sun.com* to *john.doe@isp.net*.
`suppressEmailError`	(cis, 0-1, {mta, admin}) Suppress delivery of error messages to senders. If missing or FALSE, errors are sent back to the sender. If TRUE then errors are not sent back to the sender or to the address specified in `errorsTo`.

TABLE 3-24 Optional `inetMailGroup` Attributes *(Continued)*

Attribute	Description
`mailDeliveryFile`	(ces, 0-many, {mta, admin}) Fully qualified path of a file name to which all messages submitted to this distribution list are appended. This path is on the local file system of the `mailHost` of this distribution list.
`mailDeliveryOption`	(cis, 0-many, {mta, admin}) Specifies one or more delivery options for inbound email to a designated recipient. While inbound messages can be delivered into multiple message stores, message access servers can read messages from only one of them (the mail store from which messages are read is specified using the `mailFolderMap` attribute). The Message Transfer Agent uses this attribute to determine the targets of message delivery for all messages submitted to this distribution list. The attribute is also used by the `inetMailUser` object class. The value of this attribute can take one of a specified set of options; the subset valid for distribution lists are described as follows: • mailbox – Applies only to the `inetMailUser` object class. • shared – Deliver mail to a shared mailbox in the Sun Message Store. This is used for setting up a shared mailbox for a distribution list. Access to the shared mailbox is enabled for those distribution list members whose `mailhost` attribute is the same as the `mailhost` attribute of the list. All other members of the list receive a copy of the submitted messages in their incoming mailbox. • native – Applies only to the `inetMailUser` object class. • autoreply – Applies only to the `inetMailUser` object class. • program – Deliver mail to a program. For security reasons, the value of this attribute is restricted to authorized programs. The list of such authorized programs may only be modified by the email system administrator; values are specified via the `mailDeliveryProgramInfo` attribute. The `program` option is also valid for the `inetMailUser` object class. • forward – Applies only to the `inetMailUser` object class. • file – Append incoming mail to a file. For this option to have any effect, `mailDeliveryFile` must point to a valid file, accessible from the local machine, for which the message transfer agent has write privileges. The file option is also valid for the `inetMailUser` object class. MTAs must be able to parse options other than those above, although a particular MTA may not be able to support such options. This is so that vendors may use attribute values other than those specified above. In such cases, it is recommended that the name be prefixed with a vendor-specific name, such as a stock symbol.
`mailProgramDeliveryInfo`	(ces, 0-many, {mta, admin}) Specifies one or more programs to which inbound messages will be delivered if the `mailDeliveryOption` contains a value of `program`. If the `mailDeliveryOption` does not contain a value of `program,` this attribute is ignored. Valid program names are defined as part of MTA configuration and the programs are installed on the server by the system administrator(s).

Mail List Administration Attributes

The following defines the distribution lists attributes used by administration programs.

TABLE 3-25 Optional `inetMailGroup`: Mail List Administration Attributes

Attribute	Description
`joinable`	(cis, 0-1, {admin}) Used by administrative applications to permit members to add themselves as a member of the distribution list. Accepted values are TRUE and FALSE. Missing attribute/value pair is functionally equal to `joinable=FALSE`.
`multiLineDescription`	(cis, 0-many, {admin, client}) Multi-line description of the distribution list.
`seeAlso`	(dn, 0-many, {admin, client}) Distinguished name of an entry to consult for further information.
`expandable`	(cis, 0-1, {mta, admin}) Determines whether the distribution list is expandable or not, that is, if somebody can list the addresses of the members of the distribution list. For example, if set to TRUE, the SMTP command `expn <dl_name>` returns the RFC-822 address of the members of this distribution list. When `expandable=TRUE`, the list must be expanded on the MTA only on the mail server specified in the `mailHost` attribute.
`datasource`	(cis, 0-1, {admin}) Free-form text that describes the original source or identifier of the provisioning tool.
`inetMailGroupStatus`	cis, 0-1, {client, mta, msma, admin}) Specifies the status of a distribution list. The intent of this attribute is to allow the Internet Service Provider to temporarily suspend and re-enable the distribution list. This attribute takes one of three values. If this attribute is missing from a group entry, the semantics are the same as if the value is `active`. Supported values are: • `active` • `inactive` • `deleted`

Mail Restriction Attributes

Several `inetMailGroup` attributes are key to determining who can submit messages to the distribution list. This proposal allows restrictions based on domains and addresses. One may call out the list of authorized domains/submitters or unauthorized domains/submitters.

Attributes that restrict who can submit messages to the list fall in two categories:

- authorized – Users/domains who are explicitly allowed to submit messages to the distribution list.

- unauthorized – Users / domains who are explicitly disallowed to submit messages to the distribution list.

Additionally, by specifying a moderator, the MTA can be directed to deliver submitted messages only to the moderators, unless the message is submitted by one of the moderators, in which case it is delivered to all distribution list members.

A distribution list that does not have `authorizedDomain`, `unauthorizedDomain`, `authorizedSubmitter,` and `unauthorizedSubmitter` attributes in the LDAP entry for the distribution list is treated as an unrestricted list and anybody can submit messages to this list.

If any of the `authorizedDomain`, `unauthorizedDomain`, `authorizedSubmitter,` and `unauthorizedSubmitter` attributes appear in the distribution list LDAP entry, the list is considered a restricted distribution list.

The following precedence rules are followed by the MTA when deciding whether it should accept the message for further processing or not (`From:` address is used in all the rules when looking for match):

- if `unauthorizedDomain` exists in the LDAP entry, then sender's domain must not match the domain(s) listed in the `unauthorizedDomain` attribute.
- if `authorizedDomain` attribute exists in the LDAP entry, then sender's domain must match the domain(s) listed in the `authorizedDomain` attribute.
- if `unauthorizedSubmitter` attribute exists in the LDAP entry, the sender's address must not match either the `mail` attribute or `rfc822MailAlias` attribute of any DN listed in the form of an *ldap:///<DN>* address and must not match the RFC-822 address listed in the form of a *mailto:<RFC-822>* address.
- if `authorizedSubmitter` attribute exists in the LDAP entry, the sender's address must match either the `mail` attribute or `rfc822MailAlias` attribute of any DN listed in the form of an *ldap:///<DN>* address and must not match the RFC-822 address listed in the form of a *mailto:<RFC-822>* address.

TABLE 3-26 Optional `inetMailGroup`: Mail Restriction Attributes

Attribute	Description
`moderator`	(ces, 0-many, {mta, admin}) Address of the moderator(s) of this distribution list. All messages submitted to this distribution list are delivered to the moderator(s) listed in directory entry. The moderator(s) then resubmits messages to the list for them to be delivered to the list members. The `From:` header of the resubmitted message must contain one of the addresses listed in the moderator(s) list. If the listed moderator is a distinguished name then the `From:` address must match the value of `mail` or `rfc822MailAlias` attribute of the LDAP entry specified by the DN.

TABLE 3-26 Optional `inetMailGroup`: Mail Restriction Attributes *(Continued)*

Attribute	Description
`authorizedDomain`	(cis, 0-many, {mta, admin} Domain name from which users are authorized to post to the distribution list. The wildcard character is "*". The value of this attribute should conform to RFC-822 specification. Using the wildcard character one may optionally replace a sub-domain to authorize the entire DNS hierarchy below a given top or sub-domain. A distribution list entry with an empty `authorizedDomain` allows senders from all domains to post messages to the list, except if they are called out in the following attributes: `unauthorizedDomain`, `authorizedSubmitter`, or `unauthorizedSubmitter`.
`authorizedSubmitter`	(ces, 0-many, {mta, admin}) List of all addresses authorized to submit messages to this distribution list. An open list does not restrict submissions to the list and does not contain a list of authorized / unauthorized submitters or a list of authorized / unauthorized domains. This attribute specifies the list of addresses permitted to submit messages to the distribution list. The address in `From:` header must match one of the addresses listed here before the MTA will deliver the message to a list of members.
`unauthorizedDomain`	(cis, 0-many, {mta, admin}) This attribute may be used in conjunction with `unauthorizedSubmitter` to specify sender restrictions. The domain of the sender's address is compared against those in this attribute. If there are no entries in this attribute, then all domains are allowed. However, if `authorizedDomain` has a list of domains then messages from all domains other than those in the `authorizedDomain` list are rejected. The value should conform to RFC-822 specification. The wildcard character for any field in the address is "*".
`unauthorizedSubmitter`	(ces, 0-many, {mta, admin}) Specified using the notation developed in section 4.1. Addresses of users not permitted to post messages to the list. This attribute may be used in conjunction with `authorizedSubmitter` to specify sender restrictions. The sender's address is compared against those in this attribute. If there is a match then the message is rejected. If there are no entries in this attribute then all senders are allowed. However, if `authorizedSubmitter` has a list of addresses, then messages from those senders are accepted.

Membership Attributes

Several `inetMailGroup` attributes are key to determining who can submit messages to the distribution list. This proposal allows restrictions based on domains and addresses. One may call out the list of authorized domains / submitters or unauthorized domains / submitters. Additionally, the distribution list may be marked as moderated by specifying a moderator for the distribution list.

The members of an alias or distribution list are made up of the union of the users specified in the uniqueMember attribute of the groupOfUniqueNames object class as well as the rfc822MailMember and membershipFilter attributes of the inetMailGroup object class.

TABLE 3-27 Optional inetMailGroup: Membership Attributes

Attribute	Description
rfc822MailMember	(cis, 0-many, {mta, admin}) Membership of distribution list may be specified using the uniqueMember attribute of the object class groupOfUniqueNames. However, since the syntax of the uniqueMember attribute is Distinguished Name, only users who are defined in the directory would be supported. The rfc822MailMember attribute is used to define members of a distribution list that do not have LDAP entries in the directory.
membershipFilter	(ces, 0-many, {mta, admin}) This attribute allows us to specify membership in the group using an LDAP search URL. This allows the creation of a group based on search of the directory for entries that match the given filter, rather than explicitly calling out each member individually.

Internet Mail Routing Object Classes

To avoid duplicating information, the inetMailRouting object class contains the required routing information common to all Internet email recipients. This class is required for entries describing either email users (inetMailUser) or email groups (inetMailGroup).

Note the distinction between a relay Message Transfer Agent (MTA) that relays a message and a destination MTA responsible for the final delivery of a message. A relaying MTA only needs to examine the mailHost attribute to determine the destination MTA. A destination MTA examines the mail and rfc822MailAlias attributes to determine the INBOX to which the message should be delivered.

`inetMailRouting` Object Class

The `inetMailRouting` object class is used to describe the mail sorting properties of mail recipients. This is used for both user and distribution lists. The object class is defined as follows.

```
( 1.3.6.1.4.1.42.2.27.2.2.1
NAME 'inetMailRouting'
SUP top
AUXILIARY
MUST (
    mail $ mailHost
    )
MAY (
    rfc822MailAlias
    )
)
```

TABLE 3-28 Required `inetMailRouting` Attributes

Attribute	Description
`mail`	(cis, 1, {mta, client, admin}) The user or group's advertised email address in form specified by RFC-822's `addr-speck` syntax [RFC822]. The user or group may have additional mail aliases listed in the `rfc822MailAlias` attribute. The value in this attribute must be unique for all `mail` and `rfc822MailAlias` attributes in a domain.
`mailHost`	(cis, 0-1, {mta, msma, client, admin}) Host name of the user's mail server. This is the fully qualified official host name of the mail server where a user's official Inbox is located. In the case of a distribution list, this is the fully qualified host name of the MTA where the distribution list is expanded.
`rfc822MailAlias`	(cis, 0-many, {mta, msma, client, admin) Stores alternate email aliases (RFC-822 format), if any, defined for the user or distribution list. Mail to any of the listed `rfc822MailAlias` attributes of an LDAP entry will be delivered to the user or group associated with that entry. The value in this attribute must be unique for all `mail` and `rfc822MailAlias` attributes in a domain.

Object Classes for Services

SIMS services are represented in the directory by a entry defined with an `inetService` object class.

The algorithm for determining the distinguished name of this entry is to begin with an empty distinguished name (DN) and then attach Relative Distinguished Names (RDN) for each component of the domain, most significant first. Each of these RDNs is a single `AttributeTypeAndValue`, where the type is the attribute DC and the value is an IA5 string containing the domain name component. Finally, the DN gets the root suffix of the DC tree as a suffix. For example, if the root suffix of the DC tree is `o=internet` and the fully qualified DNS name of the mail server is *mail.isp.net*, the DN of the entry is `dc=mail,dc=isp,dc=net,o=internet`.

- `inetService` – attributes for describing a SIMS service. Entries with these attributes are created under the `ou=Service` container of either the DC tree or OSI tree. The naming attribute of an `inetService` node is the version of the service.

`inetService` Object Class

The `inetService` object class is used to represent state information for and about specific services. There may also be service entries under the `Service` subnode of any SIMS domain node.

The naming attribute for this object class is `inetVersion`. Nodes created with this object class are defined under the `ou=<service_tag>,ou=Service`, container of the domain node.

The following `service_tag` values are supported in this release:

- imta – Message Transfer Agent service
- msma – Message Access and message store
- admin – Administrative server
- spm – Security Policy Manager
- provisioning – Subscriber provisioning
- SUNWftp – File Transfer Protocol
- SUNWsws – Sun Web Server

```
(OID — TBD
NAME 'inetService'
SUPERIOR 'top'
STRUCTURAL
MUST (
    commonname $ inetVersion
    )
MAY (
    inetPrivateData $ mail $ userPassword
    )
)
```

TABLE 3-29 Required `inetService` Attributes

Attribute	Description
`commonName`	(cis, 1-many) The name of the service.
`inetVersion`	(ces, 1) The version of the named service represented by this entry. This is the naming attribute for this object class.
`inetPrivateData`	(ces, 1) Reserved for use by the Sun Internet Administrator to store password data for the service represented by the entry.
`mail`	(cis, 0 - many) Specifies the email address, in RFC 822 format, to be used for status messages from this service. (Alias: `preferredRfc822Originator`).
`userPassword`	(protected, 0 - many) The password for the entry represented by this object class

CHAPTER **4**

SIMS Configuration Files

The following SIMS configuration files are covered in this chapter:

The ims.cnf File

The `ims.cnf` file is the configuration file for the Sun Internet Mail Server (SIMS) Message Store and Message Access components. The `ims.cnf` file contains configuration parameters for the Message Store and Message Access utilities.

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 parameters, Message Store delivery parameters, and Message Access parameters. The parameters are described in the following sections.

Message Store Paths

TABLE 4-1 describes the parameters for the Message Store paths.

TABLE 4-1 Message Store Paths Parameters

Parameter	Description
`ims-user-root`	Path to the per-user files. The default is `/var/opt/SUNWmail/ims/user`.
`ims-index-root`	Path to the index files. The default is `/var/opt/SUNWmail/ims/index`.
`ims-data-root`	Path to the data files. The default is `/var/opt/SUNWmail/ims/data`.
`ims-hash-root`	Path to the hashing indices. This path is currently unused but must exist. The default 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 is `/var/opt/SUNWmail/ims/adm`.
`ims-shared-root`	Path to shared mailboxes. The default is `/var/opt/SUNWmail/ims/shared`.

Message Store File System

TABLE 4-2 describes the parameters for the Message Store file system.

TABLE 4-2 Message Store File System Parameters

Parameter	Description
`ims-owner`	Solaris owner of all the Message Store files. The default is `inetmail`.
`ims-init-interval`	Number of days to create at initialization. The default is 30.
`ims-augment-interval`	Number of days to create at one time. The default is 30.

Message Store Delivery

TABLE 4-3 describes the parameters for the Message Store delivery utility (`ims_master`).

TABLE 4-3 Message Store Delivery Parameters

Parameter	Description
`ims-mail-host`	The default domain for parsing an email address when no `@domain` is present. The default is `localhost`.
`ims-parse-level`	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 is 3.
`ims-quota`	Specifies whether per-user quotas are enforced. The default is `OFF` or `ON`.
`ims-default-quota`	Default quota in bytes for users. This value is used if the information is not provided in the directory. The default is 20000000.

Message Access

TABLE 4-4 describes the parameters for the Message Access utility (`imaccessd`).

TABLE 4-4 Message Access Parameters

Parameter	Description
`ims-varmail`	`ON` specifies that users can access mailboxes in the `/var/mail` format in addition to the SIMS Message Store format. The default is `OFF`. `/var/mail` is only supported in the default domain and not in hosted domains.
`ims-maxconnections`	Number of connections that can be simultaneously supported by the message access server. The default is `10000`.
`ims-proxy`	Specifies the proxy behavior of the message access server. • `OFF` specifies the proxy is disabled—local users have access. • `ON` specifies 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 is `IMAP4 IMAP4rev1`.

TABLE 4-4 Message Access Parameters *(Continued)*

Parameter	Description
`ims-bind-address`	Specifies the interface (IP address or host name) and ports that are listened to during POP and IMAP connections. The value of `ims-bind-address` is in the form: `[hostname[=domain]][(service=port1[,port2,...][:service=port3[,port4,...]...])]` This parameter can appear multiple times in the `ims.cnf` file. • *hostname* is a host name or IP address to listen to when binding sockets in the message access server. If hostname is not specified, or if the value is *, `INADDR_ANY` is used. • *domain* is the default search domain associated with the hostname and port(s). • *service* can be specified as imap, pop3, imaps, or pop3s. If no service or ports are specified, the default ports are fetched from `/etc/services`. • *port* is one or more TCP port numbers to listen to for the specified service. Specifying a port as 0 denotes that the service is not supported on that particular server.
`ims-client-lookup`	Enables or disables the reverse DNS lookup for the clients that are logged in. The valid options are DNSON or DNSOFF (default. When the value is set to DNSOFF, only the client's IP address will be displayed.
`ims-auth-timeout`	Number of seconds after the user's last POP command before the server closes the POP connection. The usual pop client behavior is to download all messages available as fast as the server can send them and disconnect immediately. Unlike IMAP, inactive POP connections usually indicate a stale network connection, which should be terminated by the server. The default is 600 seconds (the minimum recommended in RFC 1939).
`ims-pop-timeout`	Number of seconds after the user's last POP command before the server closes the POP connection. The usual pop client behavior is to download all messages available as fast as the server can send them and disconnect immediately. Unlike IMAP, inactive POP connections usually indicate a stale network connection, which should be terminated by the server. The default is 600 seconds (the minimum recommended in RFC 1939).
`ims-pop-exclusive`	Disables concurrent access to a mailbox through pop. When this value is set to ON a user logging in through pop to the server while an active session already exists that accesses the same mailbox will not be able to view or download any messages.
`ims-ldap-failover-timeout`	Number of seconds allowed to successfully bind to a given ldap server. The default value is 30 seconds.
`ims-ldap-request-timeout`	Number of seconds allowed to search for an ldap server that can be successfully opened and bound to. This value is also the timeout for the `ldap_search`. The default value is 60 seconds.

APOP Parameters

TABLE 4-5 APOP Parameters

Parameter	Description
`ims-md5auth-enable`	Turns on or off the APOP login function. ON specifies that APOP login is allowed for users with a plaintext password in LDAP. When the value is set to OFF, APOP login is not allowed. The default is OFF. This parameter must be changed manually. You cannot change it using the SIMS Administration Console.

popb4smtp Parameters

TABLE 4-6 `popb4smtp` Parameters

Parameter	Description
`ims-popb4smtp-lib`	Enables the POP3 before the SMTP mechanism. Set the value to the full path specification of `libimpopb4smtp`. For example: `ims-popb4smtp-lib:/opt/SUNWmail/lib/libimpopb4smtp.so.1` This configuration variable does not have a default value, that is, if this variable is not set, `popb4smtp` is not turned on. This parameter must be changed manually. You cannot change it using the SIMS Administration Console.
`ims-popb4smtp-timeout`	Specifies the timeout value for `popb4smtp` entries in the IMTA database. The default value is fifteen minutes. If the value is set to zero, no new entry will be made to the IMTA database for the POP3 before SMTP. The following format is used for timeout: • D or d specifies days • H or h specifies hours • M or m specifies minutes • S or s specifies seconds For example, `1d2H3m4S` specifies a time period of 1 day, 2 hours, 3 minutes, and 4 seconds. This parameter must be changed manually. You cannot change it using the SIMS Administration Console.

The sims.cnf File

The `sims.cnf` file contains configuration parameters used by more than one component in the SIMS. The configuration file consists of lines of characters in the ASCII character set, terminated by line-feed characters.

This file has three types of lines:

- Whitespace. Consists only of spaces, tabs, blank lines, and the terminating line-feed. Whitespace is ignored.
- Comment lines. The first character of a comment line is a # character. Comment lines are ignored.
- Parameter lines. Consist of a parameter name, and equal sign, and the value for the parameter. Parameter names consist of one or more alphanumeric characters (upper and lower cases permitted) and should not contain any whitespace. The parameter line must begin in column 1.

Each entry has the form:

`parameter-name=value`

If a line begins with whitespace, it must consist only of whitespace. Such a line is not a comment because it a # does not appear in column 1, and is not a parameter line because a parameter name does not start in column 1.

The following is a sample `sims.cnf` file:

```
dcRoot=o-internet
adminBindDN=uid=ISPAdministrator,dc=isp,dc=com,o=internet
loginSeparator=+
domainHostingMode=multiple
ldapServer=ldap1:888,masterldap,lastresort
```

Note – To change the information in the `sims.cnf` file, use the imedit utility to prevent concurrent updates from leaving the file in an unexpected state.

TABLE 4-7 `sims.cnf` File Parameters

Parameter	Description
`adminBindDN`	Administrative user to bind when performing administrative functions.
`dcRoot`	Search base for looking up objects in the DC tree. For example: `dcRoot=o=internet`
`defaultDomain`	Users logging in without a `loginSeparator` in their `userid` are assumed to be in this domain. The default search base for LDAP queries is determined by converting the `defaultDomain` name to the corresponding entry in the DC tree. The default is the DNS canonical name for this host, leaving off the first host name component. For a canonical name of xyz.bar.stream.com the default `defaultDomain` is bar.stream.com. For a search base of `dc=`stream`,dc=com,o=internet` an example parameter entry is: `defaultDomain=stream.com`
`ldapServer`	Specifies a comma-separated list (no whitespace) of LDAP server locations. An LDAP server location is either `hostname=portnumber` or host name. If only a host name is specified, the port number is the default LDAP port 389. The port number is specified in decimals numbers. The default is `localhost:389`. For example: `ldapServer=localhost:389`
`logicalHostname`	The logical host name of the system. No default value. This value must be filled in during installation. For example: `logicalHostname=mail.stream.com`
`loginSeparator`	The characters use to separate the user id from the domain name when logging in to the IMAP or POP server. There are no restrictions on the login separator. The installation can select any string of non-whitespace graphic characters that is not a substring of a valid user id. No default value. If the value is not set, users cannot log in using the `uid<separator>domainname` syntax. For example: `loginSeparator=+`
`osiRoot`	Search base for looking up objects in the OSI tree. No default value. This value must be set during installation if the OSI tree will be used. For example: `osiRoot=c=us`
`spmProgramNumber`	RPC program number to use to access the SPM. For example: `spmProgramNumber=101234`
`spmServer`	Host name to use to locate the SPM RPC service. The default value is `localhost`. For example: `spmServer=spmhost.stream.com`

The imdmc.cnf File

The `imdmc.cnf` file contains configuration parameters used by the Delegated Management component in the Sun Internet Mail Server.

This file has three types of lines:

- Whitespace. Consists only of spaces, tabs, blank lines, and the terminating line-feed. Whitespace is ignored.
- Comment lines. The first character of a comment line is a # character. Comment lines are ignored.
- Parameter lines. Consist of a parameter name, and equal sign, and the value for the parameter. Parameter names consist of one or more alphanumeric characters (upper and lower cases permitted) and should not contain any whitespace. The parameter line must begin in column 1.

Each entry has the form:

`parameter-name=value`

If a line begins with whitespace, it must consist only of whitespace. Such a line is not a comment because it a # does not appear in column 1, and is not a parameter line because a parameter name does not start in column 1.

The following is a sample `imdmc.cnf` file:

```
spmServer=machine1.eng.sun.com
ws-port=80
document-root=/opt/SUNWmail/html
cgi-bin=/opt/SUNWmail/cgi-bin
```

Note – To change the information in the `imdmc.cnf` file, use the `imedit` utility to prevent concurrent updates from leaving the file in an unexpected state.

TABLE 4-8 `imdmc.cnf` File Parameters

Parameter	Description
`spmServer`	Fully qualified domain name of machine where DM server resides and runs.
`ws-port`	The Web server port number. This port number is necessary for the CGI component of the Delegated Management console to work.
`document-root`	The location of the SIMS document root directory. If the SUNWimdmr package was installed manually with a `pkgadd` on a machine with only a Web server and no other SIMS components, you will need to manually configure your Web server to point to the location of the package's html files with a symbolic link of a "sims" in your Web server `DOCUMENT ROOT` directory to `$BASEDIR/html` of the SUNWimdmi package.
`cgi-bin`	The location of the SIMS `cgi-bin` directory. If the SUNWimdmr package was installed manually with a `pkgadd` on a machine with only a Web server and no other SIMS components, you will need to manually configure your Web server to point to the location of the package's `cgi-bin` files with a symbolic link of a "sims" in your Web server `CGI-BIN` directory to `$BASEDIR/cgi-bin` of the SUNWimdmp package.

The imta.cnf File

The following is a default IMTA configuration file (`imta.cnf`) for a system not directly connected to the public internet (`stream.bridge.net`) that has a routability scope of the mail server domains (`bridge.net`).

```
! VERSION=1.2
! Modified by SIMS administration server on: Fri Mar 05 10:44:33
! PST 1999
!
! IMTA configuration file
!
! part I : rewrite rules
!
! DNS canonicalization rules.  Uncomment this line to enable DNS
! address canonicalization.
! Please refer to the SIMS documentation for details
!</etc/opt/SUNWmail/imta//dns_canonical.rules
!
```

```
! Domain Rewrite Rules.
! Uncomment this line to use domain rewrite rules
! from the configuration file instead of the domain database.
! Please refer to the SIMS documentation for details
! </tmp/newconfig/domains.rules
!
! Rules to select local users
stream.bridge.net $U%stream.bridge.net@stream.bridge.net
mailhost.eng.company $U%stream.bridge.net@stream.bridge.net
mailhost.eng $U%stream.bridge.net@stream.bridge.net
mailhost $U%stream.bridge.net@stream.bridge.net
eng.company.com $E$U%$D@stream.bridge.net
eng $U%eng.company.com@stream.bridge.net
!
! sims-ms
.sims-ms-daemon $E$U%$H.sims-ms-daemon@sims-ms-daemon
! native
.native-daemon $E$U%$H.native-daemon@native-daemon
!
! pipe
.pipe-daemon $E$U%$H.pipe-daemon@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
! tcp_default_router
! Rules for top level internet domains
</etc/opt/SUNWmail/imta//internet.rules
. $E$U%$H@tcp-daemon
!
! reprocess
reprocess $E$U%reprocess.stream.bridge.net@reprocess-daemon
reprocess.stream.bridge.net
$E$U%reprocess.stream.bridge.net@reprocess-daemon
!
! process
process $E$U%process.stream.bridge.net@process-daemon
process.stream.bridge.net $E$U%process.stream.bridge.net@process-
daemon
!
! defragment
defragment $E$U%defragment.stream.bridge.net@defragment-daemon
defragment.stream.bridge.net
$E$U%defragment.stream.bridge.net@defragment-daemon
```

```
!
! conversion
conversion $E$U%conversion.stream.bridge.net@conversion-daemon
conversion.stream.bridge.net
$E$U%conversion.stream.bridge.net@conversion-daemon
!
! bitbucket
bitbucket $E$U%bitbucket.stream.bridge.net@bitbucket-daemon
bitbucket.stream.bridge.net
$E$U%bitbucket.stream.bridge.net@bitbucket-daemon
!
! deleted
deleted-daemon $E$F%$H@deleted-daemon
.deleted-daemon $E$F%$H@deleted-daemon
!
! inactive
inactive-daemon $E$F%$H@inactive-daemon
.inactive-daemon $E$F%$H@inactive-daemon
!
! hold
hold-daemon $E$F%$H@hold-daemon
.hold-daemon $E$F%$H@hold-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
stream.bridge.net
!
! sims-ms
sims-ms queue single_job copywarnpost copysendpost postheadonly
noswitchchannel charset7 us-ascii charset8 iso-8859-1 subdirs 20
immnonurgent logging serviceall master_debug slave_debug
sims-ms-daemon
!
! native
native copywarnpost copysendpost postheadonly noswitchchannel
charset7 us-ascii charset8 iso-8859-1 subdirs 20 immnonurgent logging
serviceall
native-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
master_debug slave_debug
tcp_local-daemon stream.bridge.net
!
! tcp_default_router
tcp_default_router smtp daemon smarthost.eng.company.com
copysendpost copywarnpost postheadonly subdirs 20 immnonurgent
logging notices 1 2 4 7 master_debug slave_debug
tcp-daemon stream.bridge.net

!
! reprocess
reprocess copywarnpost copysendpost postheadonly
reprocess-daemon
!
! process
process copywarnpost copysendpost postheadonly
process-daemon
!
! defragment
defragment copywarnpost copysendpost postheadonly
defragment-daemon
!
! conversion
conversion copywarnpost copysendpost postheadonly
conversion-daemon
!
! bitbucket
bitbucket copywarnpost copysendpost postheadonly
bitbucket-daemon
!
! deleted
deleted logging
deleted-daemon
!
! inactive
inactive logging
inactive-daemon
!
! hold
hold logging
```

hold-daemon

The imta.cnf file defines several channels. The default channels defined in the sample default imta.cnf file are described in TABLE 4-9.

TABLE 4-9 The 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. The slave program /opt/SUNWmail/imta/bin/sendmail is 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
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	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 intersection of all other channel programs—they perform only operations that are shared with other channels. This is a channel queue whose contents are processed and requeued to other channels.
defragment	Provides the means to reassemble messages.
conversion	Performs body-part-by-body-part conversions on messages flowing through the IMTA.
bitbucket	Used for messages that need to be discarded.
inactive/deleted	Used to process messages for users who have been marked as inactive/deleted in the directory.
hold	Used to hold messages for users. For example, when a user is migrated from one mail server to another.

Address Rewrite Rules

Addresses are rewritten by rewrite rules in the `imta.cnf` file to convert addresses to fully qualified domain addresses and to determine their corresponding channels. The result of rewriting is a rewritten address and a routing system, that is the system to which the message is to be sent.

Address Rewrite Example

The example in this section uses a mail message and takes it through the rewrite rules.

1. A mail message arrives for `jdoe@bridge.net`.
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.
4. The `imta.cnf` file is again scanned to find a match with the domain part of the address returned by the alias database search.

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 (SIMS) 4.0. Most of these are Internet standards, published by the Internet Engineering Task Force (IETF) and approved by the Internet Activities Board (IAB). Standards for 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 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 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.
RFC 2177	Proposed Standard	Barry Leiba (IBM T.J. Watson Research Center), *IMAP4 IDLE Command*, June 1997.

SMTP and Extended SMTP

The documents listed in TABLE A-3 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 *(Continued)*

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.
RFC 2442	Information	J. Belissent, *The Batch SMTP Media Type*, November 1998.

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.
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.

TABLE A-4 Message Content and Structure *(Continued)*

Standard	Status	Description
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 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 Server Specifications

The following documents are relevant to national and international standards for directory server specifications.

Directory Server Specification

The 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 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 list documents that describe national and international telecommunications and information processing requirements.

National and International

TABLE A-9 contains 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.

TABLE A-9 National and International Information Exchange

Standard	Status	Description
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.
printableString ASN.1	International Standard	ITU-T X.680, aligned with ISO/IEC-8824-1 Abstract Syntax Notation One (ASN.1). Appears in LDAP/X.500 attribute data types. Defined jointly by the ISO, ITU-T standards bodies and have been reused in Internet RFCs and ISO, ITU-T standards.
US ASCII	National Standard	American National Standards Institute, ANSI X3.4-1986, *Coded Character Set-7-bit American National Standards 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 References

The documentation in TABLE A-10 describes Internet communications standards.

TABLE A-10 Internet References

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

ACAP Application Configuration Access Protocol. A protocol that enhances IMAP by allowing the user to set up address books, user options, and other data for universal access.

access control rules Rules specifying user permissions for a given set of directory entries or attributes.

access control list (ACL) A set of data associated with a directory that defines the permissions that users and/or groups have for accessing it.

Administration Console or Admin Console A GUI (graphical user interface) that enables you to configure, monitor, maintain, and troubleshoot the SIMS components.

address mapping See *forward address mapping* or *reverse address mapping*.

address token The address element of a rewrite rule pattern.

Administration Services A service daemon that administers components of SIMS through a GUI.

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 *MTA*.

alias An alternate name of an email address.

alias file A file used to set aliases not set in a directory, such as the postmaster alias.

APOP Authenticated Post Office Protocol. Similar to the Post Office Protocol (POP), but instead of using a plaintext password for authentication, it uses an encoding of the password together with a challenge string.

attribute The form of information stored and retrieved by the directory service. Directory information 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 that contains a given attribute or attribute value.

autoreply option file A file used for setting options for autoreply, such as vacation notices.

backbone The primary connectivity mechanism of a 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.

bang path An address for sending email using UUCP that specifies the entire route to the destination computer. It separates each host name with an exclamation point, which is also known as a bang. For example, the bang path `midearth!shire!bilbo!jsmith` would go to the `jsmith` user account on the `bilbo` host, which is reached by first going to `midearth` and then `shire`.

CA Certificate Authority. An organization that issues digital certificates (digital identification) and makes its public key widely available to its intended audience.

Certificate Authority See CA.

channel An interface with another SIMS component, another email system, or a mail user agent.

character set labels A name or label for a character set.

ciphertext Text that has been encrypted. Opposite of plaintext.

client-server model A computing model in which powerful networked computers provide specific services to other client computers. Examples include the name-server/name-resolver paradigm of the DNS and fileserver/file-client relationships such as NFS and diskless hosts.

cn LDAP alias for common name.

composition The process of constructing a message by the Mail User Agent (MUA). See also *MUA*.

configuration file A file that contains the configuration parameters for a specific component of the SIMS system.A file that contains the configuration parameters for a specific component of the SIMS system.

congestion thresholds A disk space limit that can be set by the system administrator that prevents the database from becoming overloaded by restricting new operations when system resources are insufficient.

conversion channel Converts body of messages from one form to another.

cookie Cookies are text-only strings entered into the browser’s memory automatically when you visit specific Web sites. Cookies are programmed by the Web page author. Users can either accept or deny cookies. Accepting the cookies allows the Web page to load more quickly and is not a threat to the security of your machine.

daemon A UNIX program that is not invoked explicitly, but lies dormant waiting for some condition(s) to occur. The instigator of the condition need not be aware that a daemon is lurking (though often a program will commit an action only because it knows that it will implicitly invoke a daemon). Typical daemons are print spoolers, email handlers, and schedulers that start up another process at a designated time or condition.

data store A store that contains directory information, typically for an entire directory information tree.

DC tree Domain Component tree. A directory information tree that mirrors the DNS network syntax. An example of a distinguished name in an DC tree is: `cn=billbob,dc=bridge,dc=net,o=internet`

defragmentation The Multiple Internet Mail 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*.

delegated administrator A person who has the privileges to add, modify, delete, and search for group or user entries at a specified hosted domain.

Delegated Management Console A Web browser-based software console that allows delegated administrators to add and modify users and groups to a hosted domain. Also allows end users to change their password, set message forwarding rules, set vacation rules, and list distribution list subscriptions.

delegated management server A daemon program that handles access control to the directory by hosted domains.

denial of service attack A situation in which 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.

dereferencing an alias Specifying, in a bind or search operation, that a directory service translate an alias distinguished name to the actual distinguished name of an entry.

destination channel The last element of a host/domain rewrite rule, in whose queue a message should be placed in for delivery.

directory cache A temporary storage of information that has been retrieved from the directory.

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. Also called the *IMTA directory cache*.

directory information tree The tree-like hierarchical structure in which directory entries are organized. Also called a DIT. DITs can be organized along the DNS (DC trees) or Open Systems Interconnect networks (OSI trees).

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.

directory synchronization Because information stored in the directory service is updated as new entries are added, modified and deleted, the information in the IMTA directory cache must be periodically updated with the current information in the directory service. This process is called directory synchronization. Sometimes called a `dirsync` in reference to the `imta dirsync` command.

dirsync option file A file used to set options for the `dirsync` program that 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 comma-separated sequence of attributes and values that specify the unique location of an entry within the directory information tree; often abbreviated as DN.

distribution list A list of email addresses (users) that can be sent a message by specifying one email address. Also called a group.
See also *expansion, member, moderator, owner,* and *alias.*

distribution list 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.*

DIT See *directory information tree.*

DN Distinguished name.

dn LDAP alias for distinguished name.

DNS Domain Name System. A distributed name resolution software that allows computers to locate other computers on a UNIX network or the Internet by domain name. DNS servers provide a distributed, replicated, data query service for translating host names into Internet addresses.

DNS database A database of domain names (host names) and their corresponding IP addresses.

domain A group of computers whose host names share a common suffix, the domain name. Syntactically, an Internet domain name consists of a sequence of names (labels) separated by periods (dots), for example, `tundra.mpk.ca.us`.

domain quota The amount of space, configured by the system administrator, allocated to a domain for email messages.

domain rewriting rules See *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.

dsservd A daemon that operates that accesses the database files that hold the directory information, and communicates with directory clients using the LDAP protocol.

EMAPI Extended MAPI Service Provider. Transparently turns Microsoft Exchange client into an Internet standard IMAP/LDAP client. See also *IMAP*, *LDAP*.

encryption Scrambling the contents of a message so that its contents cannot be read without the encryption, or code key.

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 protocol.

expander Part of an electronic mail delivery system that 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 (for example, *hacks@somehost.edu*) and the mail exploder takes care of delivery to the individual mailboxes in the list. Also called mail exploders.

expansion This term applies to the 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 marking a message for deletion and then permanently removing it from you INBOX.

external channel An interface between the IMTA and either another SIMS component or another component outside the SIMS email system.

failover The automatic transfer of a computer service from one system to another to provide redundant backup.

Filesharing Transport This type of transport moves messages between the UNIX operating system and the PC running a client through a shared file system available to both platforms. When a channel is configured to use filesharing transport, the shared directory to use for the file exchange must be specified.

firewall A dedicated gateway machine with special security precautions used to service outside network, especially Internet, connections and dial-in lines. The idea is to protect a cluster of more loosely administered machines hidden behind the firewall from unwanted entry from outside the firewall.

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 that 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.

FQDN See fully qualified domain name.

fragmentation The Multiple Internet Extensions (MIME) feature that allows the breaking up of a large message into smaller messages. See also *defragmentation*.

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*.

fully qualified domain name The full name of a system, consisting of its local host name and its domain name. For example, `class` is a host name and `class.sun.edu` is an fully qualified domain name. A fully qualified domain name should be sufficient to determine a unique Internet address for any host on the Internet. The same naming scheme is also used for some hosts that are not on the Internet, but share the same name-space for electronic mail addressing. A host that does not have a fully qualified domain name must be addressed using a bang path.

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.

group Same as a distribution list.

group folders Contain folders for shared and group folders. See *shared folder.*

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.

hosted domain An email domain that is outsourced by an ISP. That is, the ISP provides email domain hosting for an organization by operating and maintaining the email services for that organization. A hosted domain shares the same SIMS host with other hosted domains. In earlier LDAP-based email systems, a domain was supported one or more email server hosts. With SIMS, many domains can be hosted on a single server. Hosted domains are also called *virtual hosted domains* or *virtual domains.*

host name The logical name assigned to a computer. On the Web, most hosts are named `www`; for example, `www.mycompany.com`. If a site is composed of several hosts, they might be given different names such as `support.mycompany.com` and `sales.mycompany.com`. `support` and `sales` are the host names, `mycompany` is the subdomain name, and `com` is the top-level domain name.

IMAP4 Internet Message Access Protocol. IMAP4 provides advanced disconnected mode client access.

IMTA Internet Message Transfer Agent. IMTA routes, transports, and delivers Internet Mail messages within the email system.

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 protocol address A 32-bit address assigned to hosts using TCP/IP. Also called the *IP address* and *Internet address.*

invalid user An error condition that occurs during message handling. The message store sends a communication to the IMTA, and the message store deletes its copy of the message. The IMTA bounces the message back to the sender and deletes its copy of the message

ISP Internet Service Provider. A company that provides Internet services to its customers including email, electronic calendaring, access to the World Wide Web, and Web hosting.

job controller An IMTA daemon responsible for scheduling message delivery. 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.

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 referrals An LDAP entry that consists of a symbolic link (referral) to another LDAP entry. An LDAP referral consists of an LDAP host and a distinguished name. LDAP referrals are often used to reference existing LDAP data so that this data does not have to be replicated. They are also used to maintain compatibility for programs that depend on a particular entry that may have been moved.

LDAP Server A software server that maintains an LDAP directory and services queries to the directory. The Sun Directory Services and the Netscape Directory Services are implementations of an LDAP Server.

LDAP server failover A backup feature for LDAP servers. If one LDAP server fails, the system can switch over to another LDAP server.

LDAP filter A way of specifying a set of entries, based on the presence of a particular attribute or attribute value.

LDBM LDAP data base manager.

LDIF LDAP Data Interchange Format. A data format used to represent LDAP entries in text form.

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 also *folder*.

managed object A collection of configurable attributes, for example, a collection of attributes for the directory service.

mapping tables Two column tables that 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 that 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 submission The client Mail User Agent (MUA) transfers a message to the mail server and requests delivery.

MIB Management Information Base. A collection of objects that can be accessed using a network management protocol. See also *SMI*.

MIME Multipurpose Internet Mail Extensions. A format for defining email message content.

moderator A person who first receives all email addressed to a distribution list before A) forwarding the message to the distribution list, B) editing the message and then forwarding it to the distribution list, or C) not forwarding 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. See also *IMTA*.

MUA Mail User Agent. The client applications invoked by end users to read, submit, and organize their electronic mail.

mx record Mail Exchange Record. A DNS resource record stating a host that can handle electronic mail for a particular domain.

name resolution The process of mapping an IP address to the corresponding name. See also *DNS*.

namespace The space from which an object name is derived and understood. Files are named within the file name space; domain components are named within the domain namespace.

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 that 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`

NIS A distributed network information service containing key information about the systems and the users on the network. The NIS database is stored on the master server and all the replica or slave servers.

NIS+ A distributed network information service containing hierarchical information about the systems and the users on the network. The NIS+ database is stored on the master server and all the replica servers.

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.

notary messages Text messages sent by the MTA to an email sender indicating delivery or non-delivery status of a sent message.

object class A template specifying the kind of object the entry describes and the set of attributes it contains. For example, SIMS specifies an `emailPerson` object class that has attributes such as `commonname`, `mail` (email address), `mailHost`, and `mailQuota`.

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.

option files IMTA option files contain global parameters used to override default values of parameters that apply to IMTA as a whole, such as sizes for various tables into which various configuration and alias files are read.

OSI tree A directory information tree that mirrors the Open Systems Interconnect network syntax. An example of a distinguished name in an OSI tree would be `cn=billt,o=bridge,c=us.`

ou LDAP alias for `organizationalUnit.`

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 that performs delivery of messages by 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.

plaintext Unencrypted readable text. The opposite of cypher text

plaintext authentication Authentication that occurs by sending passwords over the network in plaintext. Considered a security problem since plaintext passwords can be easily captured over a network.

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 directory service.

protocol A formal description of messages to be exchanged and rules to be followed for two or more systems to exchange information.

provisioning The process of adding, modifying or deleting entries in the SIMS directory service. These entries include users and groups.

provisioning commands SIMS commands that provide provisioning functions. These commands are prefaced with `imadmin`.

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 permanently removing messages that have been deleted and 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*.

quota See *user quota*.

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*.

relative distinguished name The final attribute and its value in the attribute and value sequence of the distinguished name. See also *distinguished name*.

relaying A message is passed from one mail server to another mail server.

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 in which IMTA channel queue the message should be placed.

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. Refer to *http://www.imc.org/rfcs.html* for information.

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 that 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.

safe file system A 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 Veritas File System, VxFS.

schema A set of rules that sets the parameters of the data stored in a directory. It defines the type of entries, their structure and their syntax.

sendmail This program acts as a mail transport agent for Solaris software. It is responsible for routing mail and resolution of mail addresses.

shared folder or shared mailbox
: *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.

SIMS administrator
: An individual who has a valid log in and password for the SIMS Admin Console. This person can also use this log in and password to execute the provisioning CLIs.

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*.

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.

slave program
: A channel program that accepts transfers initiated by another interface.

smart host
: The mail server in a domain to which other mail servers forward messages if they do not recognize the recipients.

SMTP
: Simple Mail Transfer Protocol. The Internet electronic mail protocol. Defined in RFC 821, with associated message format descriptions in RFC 822.

SMTP Dispatcher
: A multithreaded connection dispatching agent that 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.

sn
: LDAP alias for `surname`.

SNMP
: Simple Network Management Protocol. The network management protocol of choice for TCP/IP-based internets.

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
: Sun Microsystems' implementation of an LDAP directory server. Provides storage of, and access to, user profiles, distribution lists, and other SIMS information. The Sun Directory Services is one of the three main SIMS components along with the IMTA and MS/MA.

Sun Internet Mail Server An enterprise-wide, open-standards based, scalable electronic message-handling system.

Sun Message Store The server from which mail clients retrieve and submit messages.

SSL Secure Sockets Layer is an open, non-proprietary security protocol for authenticated and encrypted communication between clients and servers.

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.

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.

uid User identification. A unique string identifying a user to a system. Also referred to as a userid.

unsafe file system A file system that 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.

upper reference Indicates the directory server that holds the naming context above your directory server's naming context in the directory information tree (DIT).

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, and so on.

user folders A user's email mailboxes.

user quota The amount of space, configured by the system administrator, allocated to a user for email messages.

user redirection The remote IMTA cannot accept mail for the recipient, but can reroute the mail to a mail server that can accept it.

UUCP UNIX to UNIX Copy Program. A protocol used for communication between consenting UNIX systems.

valid user A condition that occurs during message handling. After the message store sends a communication to the IMTA, the IMTA deletes its copy of the message and it is now the message store's responsibility.

`/var/mail` The UNIX version 7 "`From`" delimited mailbox as implemented in the Solaris operating system.

virtual hosted domains or virtual domains See *hosted domains*.

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.

Index

SYMBOLS

C

D

E

F

G

H

I

N

O

S

T

U

V

W

X