Oracle® Communications Messaging Server

System Administrator's Guide

Release 8.0

July 2015

ORACLE

Oracle Communications Messaging Server System Administrator's Guide, Release 8.0

Contents

Chapter 1. Preface

This guide explains how to administer Oracle Communications Messaging Server and its accompanying software components. **This guide applies to both Messaging Server 7 and Messaging Server 8.** Where appropriate, features or changes that were introduced in a specific release are indicated.

Audience

This document is intended for system administrators whose responsibility includes Messaging Server. This guide assumes you are familiar with the following topics:

- Messaging protocols
- Oracle Directory Server Enterprise Edition and LDAP
- System administration and networking
- General deployment architectures

Related Documents

For more information, see the following documents in the Messaging Server documentation set:

- *Messaging Server Installation and Configuration Guide*: Provides instructions for installing and configuring Messaging Server.
- *Messaging Server Administration Reference*: Provides command-line utility and configuration file information about Messaging Server.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Chapter 2. Handling sendmail Clients

Handling sendmail Clients

If users (or system utilities, for example, `cron`) send messages through `sendmail` clients, you can configure Messaging Server to work with those clients over protocol. Users can continue to use the UNIX `sendmail` client.

To create compatibility between `sendmail` clients and Messaging Server, you can create and modify a `sendmail` configuration file.

Each time a new `sendmail` patch is applied to your system, you will need to modify the `submit.cf` file as described in To Create the sendmail Configuration File on Oracle Solaris 9 Platforms.

Prior to Messaging Server 6.0, `sendmail` integration was accomplish by replacing the OS-provided `/usr/lib/sendmail` binary with a component of the Messaging Server product. Starting with Messaging Server 6.0, this replacement is no longer necessary. Therefore, if you are upgrading from Messaging Server 5 to Messaging Server 6.0 or later, you may need to obtain the proper version of the `/usr/lib/sendmail` binary from the current `sendmail` patch.

On Oracle Solaris 9 platforms, `sendmail` is no longer a `setuid` program. Instead, it is a `setgid` program.

To Create the sendmail Configuration File on Oracle Solaris 8 Platforms

1. Find the file `main-v7sun.mc` file in directory `/usr/lib/mail/cf` and create a copy of this file. In the example in this section, a copy called `sunone-msg.mc` is created.
2. In the `sunone-msg.mc` file, add the following lines before the `MAILER` macros:

   ```
   FEATURE(`nullclient', `smtp:rhino.west.sesta.com')dnl
   MASQUERADE_AS(`west.sesta.com')dnl
   define(`confDOMAIN_NAME', `west.sesta.com')dnl
   ```

   `rhino.west.sesta.com` is the localhost name and `west.sesta.com` is the default email domain as described in the discussion on creating the initial Messaging Server runtime configuration in *Unified Communications Suite Installation and Configuration Guide*. In an HA environment, use the logical host name. See Configuring Messaging Server for High Availability for more information about logical hostnames for high availability.

3. Compile the `sunone-msg.mc` file:

   ```
   /usr/ccs/bin/make sunone-msg.cf
   ```

   The `sunone-msg.mc` will output `sunone-msg.cf`.

4. Make a backup copy of the existing `sendmail.cf` file located in the `/etc/mail` directory.
   a. Copy and rename `/usr/lib/mail/cf/sunone-msg.cf` to `sendmail.cf` file.
   b. Move the new `sendmail.cf` file to the `/etc/mail` directory.

To Create the sendmail Configuration File on Oracle Solaris 9 Platforms

1. Find the file `submit.mc` file in directory `/usr/lib/mail/cf` and create a copy of this file. In the example in this section, a copy called `sunone-submit.mc` is created.
2. Change the following line in the file `sunone-submit.mc`:

   ```
   FEATURE(`msp')dn
   ```

   to

   ```
   FEATURE(`msp', `rhino.west.sesta.com')dnl
   ```

   where `rhino.west.sesta.com` is the localhost name. `rhino.west.sesta.com` is the localhost name and `west.sesta.com` is the default email domain as described in the discussion on creating the initial Messaging Server runtime configuration in *Unified Communications Suite Installation and Configuration Guide*. In an HA environment, use the logical host name. See Configuring Messaging Server for High Availability for more information about logical hostnames for high availability.
3. Compile the `sunone-submit.mc` file:

   ```
   /usr/ccs/bin/make sunone-submit.cf
   ```

   The `sunone-submit.mc` will output `sunone-submit.cf`.
4. Make a backup copy of the existing `submit.cf` file in the `/etc/mail` directory.
   a. Copy and rename `/usr/lib/mail/cf/sunone-submit.cf` file to `submit.cf` file.
   b. Move the new `submit.cf` file to the `/etc/mail` directory.

Chapter 3. Configuring Messaging Server for High Availability

Configuring Messaging Server for High Availability

For information about configuring Messaging Server 7.0 for high availability, see the following chapter in the *Messaging Server 6.3 Administration Guide*: **Configuring High Availability**.

The preceding link provides the information you need to configure the Veritas Cluster Server or Oracle Solaris Cluster high availability clustering software and prepare it for use with the Messaging Server.

> **Note: About the Legacy Documentation**
> Some information in the Messaging Server 6.3 documentation is out of date. The following sections describe information new in Messaging Server 7.0 which supersedes corresponding information in the *Messaging Server 6.3 Administration Guide*.

The remainder of this page contains the following topics:

New Supported Versions of High-Availability Software in Messaging Server 7.0

Ignore section **3.1, Supported Versions**, in the *Messaging Server 6.3 Administration Guide*.

For the latest supported versions and platforms, see the topic on high availability support in the Release Notes in *Unified Communications Suite Installation and Configuration Guide*.

New Installation Methods for Messaging Server 7.0

Ignore section **3.3.1, Cluster Agent Installation**, in the *Messaging Server 6.3 Administration Guide*.

Instead, use the following information:

A cluster agent is a Messaging Server program that runs under the cluster framework.

The Oracle Solaris Cluster Messaging Server agent is installed when you select Messaging Server Oracle Solaris Cluster HA Agent 7.0 through the Communications Suite installer.

1. Run the Communications Suite installer command:

```
./commpkg install
```

When prompted, select the Messaging Server Oracle Solaris Cluster HA Agent 7.0 software.

2. Run the Oracle Solaris Cluster HA Agent pre-configuration command:

```
cd <ms_scha_base>/bin/
./init-config
```

New Installation Paths for Messaging Server Oracle Solaris Cluster HA Agent 7.0

Wherever you see references to the Messaging Server 6.3 *msg-svr-base* path:

```
/opt/SUNWmsgsr/
```

use the Messaging Server 7.0 *msg-svr-base* path instead:

For 32-bit Messaging Server 7.0:

```
/opt/sun/comms/messaging
```

For 64-bit Messaging Server 7.0:

```
/opt/sun/comms/messaging64
```

The Messaging Server Oracle Solaris Cluster HA Agent 7.0 is by default now installed in the following path:

```
/opt/sun/comms/msg_scha
```

Installing Messaging Server Oracle Solaris Cluster HA Agent in Solaris Zones

Oracle Solaris Cluster has added support for Oracle Solaris Zones. In this scenario on Solaris 10, the Messaging Server Oracle Solaris Cluster HA agent should be installed in the global zone (and automatically installed in non-global zones). The Comms Installer will do this for you as long as you do the install in the global zone. On Solaris 11, you must install the Messaging Server Oracle Solaris Cluster HA agent in each non-global zone you need it installed.

Take the following steps to install the Messaging Server Oracle Solaris Cluster HA agent in non-global zones:

1. Run the Communications Suite installer command only in the global zone if you are running Solaris 10. If you are running Solaris 11, run the command in each zone you need it installed:

```
./commpkg install
```

When prompted, select the Messaging Server Oracle Solaris Cluster HA Agent 7.0 software. This

command installs the Messaging Server Oracle Solaris Cluster HA Agent package on global zone and all non-global zones if on Solaris 10. If on Solaris 11, run the command in each zone you need it installed.

2. Run the Oracle Solaris Cluster HA Agent pre-configuration command in the global zone only:

```
cd <ms_scha_base>/bin/
./init-config
```

Using the `useconfig` Utility

The `useconfig` utility allows you to share a single configuration between multiple nodes in an HA environment. This utility is not meant to upgrade or update an existing configuration. Note that only `useconfig` command usage has been changed in this release. All the MS HA info from the previous release is still valid.

For example, if you are upgrading your first node, you will install through the Communications Suite Installer and then configure Messaging Server. You will then failover to the second node where you will install the Messaging Server package through the Communications Suite Installer, but you will not have to run the Initial Runtime Configuration Program (configure) again. Instead, you can use the useconfig utility.

To enable the utility, run `useconfig` to point to your previous Messaging Server configuration:

```
<msg-svr-base>/sbin/useconfig <msg-svr-base>/config
```

Chapter 4. Configuring General Messaging Capabilities

Configuring General Messaging Capabilities

This information describes the general Messaging Server tasks, such as starting and stopping services and configuring directory access by using command-line utilities. Tasks specific to administering individual Messaging Server services, such as POP, IMAP, HTTP, and SMTP, are described in elsewhere.

Topics:

To Modify Your Passwords

If you set up a number of administrators with the same password during the initial Messaging Server configuration, you might want to change the passwords of those administrators.

The following table shows the parameters where default passwords are set up during initial runtime configuration and the utilities that you can use to change them. For those parameters that use the `configutil` utility, see `configutil` for complete syntax and usage.

Passwords Set in Messaging Server Initial Runtime Configuration

Parameter	Description
`local.ugldapbindcred`	Password for the Messaging Server LDAP user/group access account (`local.ugldapbinddn`) set through the `configutil` utility.
`local.service.proxy.adminpass`	Password for the Proxy Administrator account (`local.service.proxy.admin`), which is used to provide proxy authentication access to end-user mailboxes and is set through the `configutil` utility.
`local.service.http.smtpauthpassword`	Password used when `mshttpd` submits mail to the MTA. Set by initial configuration to the same password as `local.service.proxy.adminpass`.
SSL passwords for key files	Passwords that are directly set in the `sslpassword.conf` file.
Admin Account credentials	The "admin" account is both in the service administrator group by default and is a store admin by default. You are prompted for this password during initial configuration. By default, the "admin" account is used for proxy and SMTP authentication, so this password needs to match the settings for `local.service.proxy.adminpass` and `local.service.http.smtpauthpassword`.
Messaging End User Administrator	This is the LDAP user for this specific host. The `local.ugldapbindcred` entry and the "Messaging End User Administrator" actually refer to the same password, which is set both in the option and in the `userPassword` attribute for that user in the LDAP directory. The password is generated randomly by initial configuration and is only used by one single Messaging Server host to bind to the LDAP directory server to perform searches.

The following example uses the `local.service.proxy.adminpass` parameter to change the password of the Proxy Administrator account:

```
./configutil -o local.service.proxy.adminpass -v newpassword
```

Managing Mail Users, Mailing Lists and Domains

User, mailing list, and domain information is stored as entries in an LDAP directory. An LDAP directory can contain a wide range of information about an organization's employees, members, clients, or other types of individuals that in one way or another "belong" to the organization. These individuals constitute the *users* of the organization.

In the LDAP directory, the information about users is structured for efficient searching, with each user entry identified by a set of attributes. Directory attributes associated with a user can include the user's name and other identification, division membership, job classification, physical location, name of manager, names of direct reports, access permission to various parts of the organization, and preferences of various kinds.

In an organization with electronic messaging services, many if not all users hold mail accounts. Messaging Server stores copies of some account information (uid and quota in particular) on local

servers. In general, the LDAP directory is considered authoritative for account information by Messaging Server. Once account information for a mail user is present in the LDAP directory, then the mail server named in the `mailHost` attribute automatically creates that user without any additional mail server specific configuration.

Creating and managing mail users and mailing lists consists of creating and modifying user and mailing list entries in the LDAP directory. This is done using the Delegated Administrator GUI or command-line utilities, or by directly modifying the LDAP directory information.

> **Note**
> In general, the Messaging Server documentation does not describe how to directly modify the LDAP directory. Consult the Directory Server documentation for more information.

To Remove a User from Messaging Server by Using Delegated Administrator

1. Mark the user as deleted by running the `commadmin user delete` command. (See the topic on removing users, groups, and services from a domain in *Delegated Administrator System Administrator's Guide* for more information.)
2. Remove services from the user.
   A service can be a mailbox or a calendar. For the current version of Messaging Server, the program is called `msuserpurge`. For calendar services, the program is `csclean`. (See Sun Java System Calendar Server 6.3 Administration Guide for more information.)
3. Permanently remove the user, by invoking the `commadmin domain purge` command.

To Remove a Domain from Messaging Server using Delegated Administrator

1. Mark the domain as deleted by running the `commadmin domain delete` command. (See the topic on removing users, groups, and services from a domain in *Delegated Administrator System Administrator's Guide* for more information.)
2. Remove services from the users of that domain.
   A service can be a mailbox or a calendar. For Messaging Server, the program is called `msuserpurge`. For calendar services, the program is `csclean`. (See *Calendar Server System Administrator's Guide*.)
3. Permanently remove the domain, by invoking the `commadmin domain purge` command.

Starting and Stopping Services

How you stop and start Messaging Server services depends on if they are installed in an HA environment.

To Start and Stop Messaging Server Services in an HA Environment

While Messaging Server is running under HA control, you cannot use the normal Messaging Server start, restart, and stop commands to control individual Messaging Server services. For example, if you attempt a `stop-msg` in an HA deployment, the system warns that it has detected an HA setup and informs you how to properly stop the system.

The appropriate HA start, stop, and restart commands are shown in the following tables. There are no specific HA commands to individually start, restart, or stop other Messaging Server services (for example, SMTP). However, you can run a `stop-msg` *service* command to stop/restart individual servers such as `imap`, `pop` or `sched`.

Sun Cluster's finest granularity is that of an individual resource. Since Messaging Server is known to Sun Cluster as a resource, the Sun Cluster `scswitch` commands affect all Messaging Server services as a

whole.

Start, Stop, Restart in a Sun Cluster 3.0/3.1 Environment

Action	Individual Resource	Entire Resource Group
Start	`scswitch -e -j` *resource*	`sscswitch -Z -g` *resource_group*
Restart	`scswitch -n -j` *resource* `scswitch -e -j` *resource*	`scswitch -R -g` *resource_group*
Stop	`scswitch -n -j` *resource*	`scswitch -F -g` *resource_group*

Start, Stop, Restart in a Sun Cluster 3.2 Environment

Action	Individual Resource	Entire Resource Group
Start	`clrs online` *resource*	`clrg online` *resource_group*
Restart	`clrs disable` *resource* `clrs enable` *resource*	`clrg restart` *resource_group*
Stop	`clrs offline` *resource*	`clrg disable` *resource_group*

Start, Stop, Restart in Veritas 3.5, 4.0, 4.1 and 5.0 Environments

Action	Individual Resource	Entire Resource Group
Start	`hares -online` *resource* -sys *system*	`hagrp -online` *group* -sys *system*
Restart	`hares -offline` *resource* -sys *system* `hares -online` *resource* -sys *system*	`hagrp -offline` *group* -sys *system* `hagrp -online` *group* -sys *system*
Stop	`hares -offline` *resource* -sys *system*	`hagrp -offline` *group* -sys *system*

To Start and Stop Messaging Server Services in a non-HA Environment

- Start and stop services from the command line by using the following commands:
  msg-svr-base`/sbin/start-msg`
  msg-svr-base`/sbin/stop-msg`

Though you can use the command template to start and stop services individually (*msg-svr-base* `/sbin/stop-msg` *service* (where service can be `mta`, `imap`, `pop`, `store`, `http`, `ens`, `sched`, `purge`, `mfagent`, `snmp`, `mmp`, `sms`, `metermaid`, `cert`, `dispatcher`, `job_controller` or `watcher`)), do not do so except in specific tasks as described. Certain services have dependencies on other services and must be started in a prescribed order. Complications can arise when trying to start services on their own. For this reason, you should start and stop all the services together using the `start-msg` and `stop-msg` commands.

> **Note**
> You must first enable services such as POP, IMAP, and HTTP, before starting or stopping them. For more information, see Enabling and Disabling Services.

Important
If a server process crashes, other processes might hang as they wait for locks held by the server process that crashed. If you are not using automatic restart (see Automatic Restart of Failed or Unresponsive Services), and if any server process crashes, you should stop **all** processes, then restart **all** processes. This includes the POP, IMAP, and MTA processes, as well as the `stored` (message store) process, and any utilities that modify the message store, such as `mboxutil`, `deliver`, `reconstruct`, `readership`, or `upgrade`.

To Start Up, Shut Down, or View the Status of Any Messaging Services

Do not shut down individual services except in the specific tasks as described. Certain services have dependencies on other services and must be started in a prescribed order. Complications can arise when trying to start services on their own. For this reason, you should start and stop all the services together by using the `start-msg` and `stop-msg` commands.

However, when you make a configuration change that requires restart of a service, and you want to minimize service disruption, then use `stop-msg` *service* followed by `start-msg` *service*. For example, when changing an MTA option that requires a restart of the dispatcher but does not require a restart of the job_controller, it is better to run `stop-msg dispatcher; start-msg dispatcher` to avoid unnecessarily flushing the job_controller's cache. For more information, see start-msg and stop-msg.

The services must be enabled to stop or start them. See To Specify What Services Can Be Started.

To Specify What Services Can Be Started

By default the following services are started with `start-msg`:

```
#./start-msg
Connecting to watcher ...
Launching watcher ... 9347
Starting store server .... 9356
Checking store server status ..... ready
Starting purge server .... 9413
Starting imap server .... 9420
Starting pop server .... 9425
Starting http server .... 9437
Starting sched server ... 9451
Starting dispatcher server .... 9461
Starting job_controller server .... 9466
```

These can be controlled by enabling or disabling the following `configutil` parameters:

```
service.imap.enable
service.pop.enable
service.http.enable
local.smsgateway.enable
local.snmp.enable
local.imta.enable
local.mmp.enable
local.ens.enable
local.sched.enable
local.purge.enable
```

```
local.store.enable
local.metermaid.enable
local.mfagent.enable
```

Set both `service.imap.enable` and `service.imap.enablesslport` to `0` to disable IMAP. The same goes for POP and HTTP. See configutil reference for details.

Starting and Stopping a Messaging Server Running in MTA-only Mode

To start an MTA-only system, you should also start `imsched`. Before you do this, remove any scheduled jobs that are not appropriate to your installations.

`imsched` is an individual component of Messaging Server that must be started separately if you are not starting all of Messaging Server. If you start your MTA-only system by using `start-msg imta` or `start-msg mta`, then you do not run the `imsched` process.

To run messaging server in MTA mode only (no store, imap, pop, or http processes), you can either select the MTA to be only installed and configured during the Messaging Server configuration after initial install (*msg_base*`/sbin/configure`), or manually disable the message store and `mshttp` process by using the following `configutil` commands:

```
./configutil -o local.store.enable -v 0
./configutil -o service.http.enable -v 0
```

Once you have disabled HTTP and other store processes, you can then start Messaging Server by running the following command:

```
# ./start-msg
bash-3.00# ./start-msg
Connecting to watcher ...
Launching watcher ... 4034
Starting ens server ... 4035
Starting sched server ... 4036
Starting dispatcher server .... 4038
Starting job_controller server .... 4042
```

All the appropriate processes are started, including `imsched` and `imta`. This way you do not have to remember to start the `sched` process.

Automatic Restart of Failed or Unresponsive Services

Messaging Server provides two processes called `watcher` and `msprobe` that transparently monitor services and automatically restart them if they crash or become unresponsive (the services hangs). `watcher` monitors server crashes. `msprobe` monitors non-responsive server processes by checking their response times. When a server fails or stops responding to requests, it is automatically restarted. The following table shows the services monitored by each utility.

Services Monitored by watcher and msprobe

watcher (crash)	msprobe (unresponsive hang)
IMAP, POP, HTTP, job controller, dispatcher, message store (`stored`), `imsched`, MMP. (LMTP/SMTP servers are monitored by the dispatcher and LMTP/SMTP clients are monitored by the job_controller.) The watcher also monitors all processes that access the message store in such a way that they could hold outstanding message store locks when they crash. This includes `ims_master`, `lmtp_server`, and store utilities.	IMAP, POP, HTTP, cert, job controller, message store (`stored`), `imsched`, ENS, LMTP, SMTP

Setting `local.watcher.enable=on` (default) monitors process failures and unresponsive services and logs error messages to the `default` log file indicating specific failures. To enable automatic server restart, set the `configutil` parameter `local.autorestart` to `yes`. By default, this parameter is set to `no`.

If any of the message store services fail or freeze, all message store services that were enabled at start-up are restarted. For example, if `imapd` fails, at the least, `stored` and `imapd` are restarted. If other message store services were running, such as the POP or HTTP servers, then those are restarted as well, whether or not they failed.

Automatic restart also works if a message store utility fails or freezes. For example, if `mboxutil` fails or freezes, the system automatically restarts all the message store servers. However, it does not restart the utility. `msprobe` runs every 10 minutes. Service and process restarts are performed once within a 10-minute period (and are configurable by using `local.autorestart.timeout`). If a server fails more than once during this designated period of time, then the system stops trying to restart this server. If this happens in an HA system, Messaging Server is shut down and a failover to the other system occurs.

Whether or not `local.autorestart` is set to `yes`, the system still monitors the services and sends failure or non-response error messages to the console and *msg-svr-base*`/data/log/watcher` listens to port 49994 by default, but this is configurable with `local.watcher.port`.

A watcher log file is generated in *msg-svr-base*`/data/log/watcher`. This log file is not managed by the logging system (no rollover or purging) and records all server starts and stops. An example log is shown below:

```
watcher process 13425 started at Tue Oct 21 15:29:44 2003

Watched 'imapd' process 13428 exited abnormally
Received request to restart:  store imap pop http
Connecting to watcher ...
Stopping http server 13440 .... done
Stopping pop server 13431 ... done
Stopping pop server 13434 ... done
Stopping pop server 13435 ... done
Stopping pop server 13433 ... done
imap server is not running
Stopping store server 13426 .... done
Starting store server .... 13457
checking store server status ...... ready
Starting imap server ..... 13459
Starting pop server ....... 13462
Starting http server ...... 13471
```

See Monitoring Using msprobe and watcher Functions for more details on how to configure this feature.

msprobe is controlled by imsched. If imsched crashes, this event is detected by watcher and triggers a restart (if autorestart is enabled). However, in the rare occurrence of imsched hanging, you need to kill imsched with a kill *imsched_pid* command, which causes the watcher to restart it.

Automatic Restart in High Availability Deployments

The following table shows the following configutil parameters to be set for automatic restart in high availability deployments:

HA Automatic Restart Parameters

Parameter	Description/HA Value
local.watcher.enable	Enable watcher on start-msg startup. Default is yes.
local.autorestart	Enable automatic restart of failed or frozen (unresponsive) servers including IMAP, POP, HTTP, job controller, dispatcher, and MMP servers. Default is No.
local.autorestart.timeout	Failure retry time-out. If a server fails more than once during this designated period of time, then the system will stop trying to restart this server. If this happens in an HA system, Messaging Server is shutdown and a failover to the other system occurs. The value (set in seconds) should be set to a period value longer than the msprobe interval. (See local.schedule.msprobe below). Default is 600.
local.schedule.msprobe	msprobe run schedule. A crontab style schedule string (see the Expire and Purge configutil Log and Scheduling Parameters) table. Default is 5,15,25,35,45,55 * * * * lib/msprobe To disable: set local.schedule.msprobe.enable to NO.

To Schedule Automatic Tasks

Messaging Server provides a general task scheduling mechanism by using a process called imsched. It is intended for scheduling Messaging Server processes. It is enabled by setting the local.schedule.*taskname* configutil parameter. If you modify the schedule, either restart the scheduler with the command stop-msg sched and start-msg sched, or refresh the scheduler process (refresh sched).

This parameter requires a command and a schedule on which to execute the command. The format is as follows:

configutil -o local.schedule.*taskname* -v "*schedule*"

taskname is a unique name for this command/schedule combination.

schedule has the format:

minute hour day-of-month month-of-year day-of-week command args

command args can be any Messaging Server command and its arguments. Paths can be relative to *msg-svr-base* or absolute paths. See Pre-defined Automatic Tasks for relative path examples.

minute hour day-of-month month-of-year day-of-week is the schedule for running the command. It follows the UNIX crontab time format.

The values are separated by a space or tab and can be 0-59, 0-23, 1-31, 1-12 or 0-6 (with 0=Sunday)

respectively. Each time field can be either an asterisk (meaning all legal values), a list of comma-separated values, or a range of two values separated by a hyphen. Days can be specified by both day of the month and day of the week and both will be required if specified. For example, setting the 17th day of the month and Tuesday will only run the command on the 17th day of a month when it is Tuesday. see the Expire and Purge configutil Log and Scheduling Parameters) table.

If you modify scheduler, either restart the scheduler with the command `stop-msg sched` and `start-msg sched`, or refresh the scheduler process:

```
refresh sched
```

To disable a scheduled task, run the following:

```
./configutil -o local.schedule.<taskname>.enable -v no
./refresh sched
```

Scheduler Examples

Run `imexpire` at 12:30am, 8:30am, and 4:30pm:

```
./configutil -o local.schedule.expire -v "30 0,8,16 * * * bin/imexpire"
```

Run `imsbackup` Monday through Friday at midnight (12AM):

```
./configutil -o local.schedule.msbackup -v "0 0 * * 1-5 bin/imsbackup -f
backupfile /primary"
```

Pre-defined Automatic Tasks

At installation, Messaging Server creates, schedules and enables the following set of pre-defined automatic tasks:

The following automatic tasks are set and enabled for the message store:

```
local.schedule.expire = "0 23 * * * bin/imexpire"
local.schedule.expire.enable = 1
local.schedule.snapshotverify =
"1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,5
* * * * bin/imdbverify -m"
local.schedule.snapshotverify.enable = 1
```

The following automatic tasks are set and enabled for the MTA:

```
local.schedule.purge = "0 0,4,8,12,16,20 * * * bin/imsimta purge -num=5"
local.schedule.purge.enable = 1
local.schedule.return_job = "30 0 * * * lib/return_job"
local.schedule.return_job.enable = 1
```

The following automatic tasks are set and enabled for the message store:

```
local.schedule.msprobe = "5,15,25,35,45,55 * * * * lib/msprobe"
```

To Configure a Greeting Message

Messaging Server enables you to create an email greeting message to be sent to each new user.

To Create a New User Greeting

- To create a new-user greeting use the command line:

  ```
  configutil -o gen.newuserforms -v <Message>
  ```

  Where *Message* must contain a header (with at least a subject line), followed by $$, then the message body. The $ represents a new line.
  For example, to enable this parameter, you can set the following configuration variables:

  ```
  ./configutil -o gen.newuserforms -v 'Subject: Welcome!! $$ Sesta.com
  welcomes you to the premier Internet experience in Dafandzadgad!'
  ```

  Depending on the shell that you are using, it might be necessary to append a special character before $ to escape the special meaning of $. ($ is often the escape character for the shell.)

To Set a Per-Domain Greeting Message

Whenever you create a new hosted domain, create per-domain greeting messages for your supported languages. If this is not done, the generic greeting message set by `gen.newuserforms` is sent.

You can set a greeting message for new users in each domain. The message can vary depending on the user's, the domain's, or the site's preferred language. This is done by setting the `mailDomainWelcomeMessage` attribute in the desired LDAP domain entry. The attribute syntax is as follows:

`mailDomainWelcomeMessage;lang-`*user_prefLang*
`mailDomainWelcomeMessage;lang-`*domain_prefLang*
`mailDomainWelcomeMessage;lang-`*gen.sitelanguage*

The following example sets the domain welcome message for English:

```
mailDomainWelcomeMessage;lang-en: Subject: Welcome!! $$Welcome to the mail
system.
```

The following example sets the domain welcome message for French:

```
mailDomainWelcomeMessage;lang-fr: Subject: Bienvenue!! $$Bienvenue a
siroe.com!
```

Using these examples, assume the following:

- The domain is `siroe.com`.

- A new user belongs to this domain.
- The user's preferred language is French as specified by the LDAP attribute `preferredlanguage`.
- The `siroe.com` domain has the above English and French welcome messages available.
- The site language is `en` as specified by `gen.sitelanguage`.

For a list of supported locales and their language value tag, see *Directory Server Reference*.

When users log in for the first time, they will receive the French greeting. If the French welcome message isn't available, they will get the English greeting.

Greeting Message Theory of Operations

Greeting messages can be set by both the LDAP attribute `mailDomainWelcomeMessage` and the `configutil` parameter `gen.newuserforms`. The order in which a message will get chosen, with the top one having the highest preference, is shown below:

`mailDomainWelcomeMessage;lang-`*user_prefLang*
`mailDomainWelcomeMessage;lang-`*domain_prefLang*
`mailDomainWelcomeMessage;lang-`*gen.sitelanguage*
`mailDomainWelcomeMessage`
`gen.newuserforms;lang-`"$user-*prefLang*"
`gen.newuserforms;lang-`"$domain-*prefLang*"
`gen.newuserforms;lang-`"$gen.*sitelanguage*"
`gen.newuserforms`

The algorithm works as follows: if there are no domains (or there are, but there is no per domain welcome message provisioned for them), a welcome message is configured with the `gen.newuserforms` parameter, if specified. If a user has a preferred language (set with the `preferredlanguage` LDAP attribute) and `gen.newuserforms;lang-`*user_prefLang* is set, the user will receive that welcome message at the time of their first log in to the server. If `gen.newuserforms;lang-`*gen.sitelanguage* is set, and `preferredlanguage` is not set, but the site language is set (using `gen.sitelanguage` parameter), user will receive that message. If no language tag parameter is set and a untagged `gen.newuserforms` is set, then that message will be sent to the user. If none of the values are set, user will not receive any welcome message.

If the user is in a domain, then similar to the discussion above, the user might receive one of `mailDomainWelcomeMessage;lang-xx`, depending on which one is available in the list and in the order given.

Example: Domain is `siroe.com`. The domain preferred language is German (de). But the new user in this domain has preferred language of Turkish (tr). Site language is English. The following values are available (`mailDomainWelcomeMessage` are attributes of the domain siroe.com):

```
mailDomainWelcomeMessage;lang-fr
mailDomainWelcomeMessage;lang-ja
gen.newuserforms;lang-de
gen.newuserforms;lang-en
gen.newuserforms
```

According to the algorithm, the message sent to the user is `gen.newuserforms;lang-de`.

To Set a User-Preferred Language

Administrators can set a preferred language for the GUI and server-generated messages by setting the attribute `preferredLanguage` in the user's LDAP entry.

When the server sends messages to users outside of the server's administrative domain it does not know what their preferred language is unless it is responding to an incoming message with a preferred language specified in the incoming message's header. The header fields (`Accept-Language`, `Preferred-Language` or `X-Accept-Language`) are set according to attributes specified in the user's mail client.

If there are multiple settings for the preferred language, the server chooses the preferred language. For example, if a user has a preferred language attribute stored in the Directory Server and also has a preferred language specified in their mail client, the server chooses the preferred language in the following order:

1. The `Accept-Language` header field of the original message.
2. The `Preferred-Language` header field of the original message.
3. The `X-Accept-Language` header field of the original message.
4. The preferred language attribute of the sender (if found in the LDAP directory).

To Set a Domain Preferred Language

A domain preferred language is a default language specified for a particular domain. For example, you can specify Spanish for a domain called `mexico.siroe.com`. Administrators can set a domain preferred language by setting the attribute `preferredLanguage` in the domain's LDAP entry.

To Specify a Site Language

You can specify a default site language for your server as follows. The site language is used to send language-specific versions of messages if no user preferred language is set.

- *Command Line*: Specify a site language as follows:
  `configutil -o gen.sitelanguage -v` *value*
  where *value* is one of the local supported languages. See the Directory Server documentation for a list of supported locales and the language value tag.

Encryption Settings

This is described in To Enable SSL and Selecting Ciphers, which also contains background information on all security and access-control topics for Messaging Server.

Setting a Failover LDAP Server

It is possible to specify more than one LDAP server for the user/group directory so that if one fails another takes over.

To Set a Failover LDAP Server

1. Set `local.ugldaphost` to the multiple replicated LDAP servers.
   `configutil -o local.ugldaphost -v "server1 server2[:port] ..."`
   For example:

```
./configutil -o local.ugldaphost -v "ldap1.sesta.com
ldap2.sesta.com:389"
```

2. If you are using a compiled MTA configuration then recompile the MTA configuration file.

```
./imsimta cnbuild
```

3. Restart Messaging Server.

```
./stop-msg
./start-msg
```

Email Security Concerns

By default, Messaging Server enables four private commands called XADR, which returns information about how an address is routed internally by the MTA as well as general channel information, XCIR, which returns MTA circuit check information, XGEN, which returns status information about whether a compiled configuration and compiled character set are in use, and XSTA, which returns status information about the number of messages processed and currently in the MTA channel queues. Releasing such information may constitute a breach of security for some sites.

Sites might want to disable these commands for the tcp_local channel by by adding the following lines to the file `tcp_local_options`, creating it if necessary.

```
DISABLE_ADDRESS=1
DISABLE_CIRCUIT=1
DISABLE_STATUS=1
DISABLE_GENERAL=1
```

Chapter 5. Enabling Single Sign-On (SSO)

Enabling Single Sign-On (SSO) in Messaging Server

> Starting with **Messaging Server 7 Update 3**, existing Messenger Express deployments that use Single Sign-On (SSO) no longer work. All versions of Convergence and more recent versions of Communications Express have their own configuration to tie into Access Manager or Trusted Circle SSO.
>
> As of **Communications Suite 7 Update 1**, Access Manager and Sun OpenSSO has been deprecated.

Single sign-on is the ability for an end user to authenticate once (that is, log on with user ID and password) and have access to multiple applications. Access Manager (formerly Identity Server) is the official gateway used for SSO for Oracle servers. That is, users must log into Access Manager to get access to other SSO configured servers.

For example, when properly configured, a user can sign in at the Access Manager login screen and have access to Messenger Express in another window without having to sign in again. Similarly, if the Calendar Server is properly configured, users can sign in at the Access Manager login screen, then have access to their Calendar in another window without having to sign in again.

Messaging Server actually provides two methods of deploying SSO. The first way is through Access Manager, the second way is through Trusted Circle technology. Using a trusted circle is the legacy method of implementing SSO. Though this method provides some features not available with Access Manager SSO, do not use it since all future development is the Access Manager. Both methods, however, are described in this information.

Topics:

Access Manager SSO for Oracle Servers

This section describes the following topics for SSO using Access Manager.

SSO Limitations and Notices

- The Messenger Express session is only valid for as long as the Access Manager session is valid. If the user logs out of Access Manager the webmail session is automatically closed (single sign-off).
- SSO Applications working together must be in the same DNS domain. (Also known as cookie domain).
- SSO Applications must have access to Access Manager verification URL (naming service).
- Browsers must have cookies.

Configuring Messaging Server to Support SSO

Four `configutil` parameters support Messaging Server SSO. Of these four, only one, `local.webmail.sso.amnamingurl` is required to enable SSO with Messaging Server. To enable SSO, set this parameter to the URL where Access Manager runs the naming service. Typically this URL is `http://`*server*`/amserver/namingservice`. Example:

```
configutil -o local.webmail.sso.amnamingurl -v
http://sca-walnut:88/amserver/namingservice
```

Note
Access Manager SSO does not look at `local.webmail.sso.enable`, which enables the older SSO mechanism. `local.webmail.sso.enable` should be left to `off` or unset, otherwise warning messages are logged about missing configuration parameters that are needed for the old SSO mechanism.

You can modify the SSO configuration parameters shown in the following table, by using the `configutil` command.

Access Manager Single Sign-On Parameters

Parameter	Description
`local.webmail.sso.amnamingurl`	The URL where Access Manager runs the naming service. Mandatory variable for single sign-on through Access Manager. Typically this URL is `http://`*server* `/amserver/namingservice`. Default: Not set.
`local.webmail.sso.amcookiename`	Access Manager cookie name. By default Access Manager saves its session handle in a cookie called `iPlanetDirectoryPro`. If it is configured to use another cookie name, then that name needs to be configured in Messaging Server as this parameter so that Messaging Server knows what to look for when doing single-sign on. Default value must not be changed if IS has default configuration. Default: `iPlanetDirectoryPro`
`local.webmail.sso.amloglevel`	AMSDK logging level. The SSO library used by Messaging Server has its own logging mechanism separate from Messaging Server. Its messages are logged in a file called `http_sso` under *msg-svr-base*`/log`. By default only messages with *info* or higher are logged, but it is possible to increase the logging level by setting the logging level to a value from 1 to 5 (1 = errors, 2 = warnings, 3 = info, 4 = debug, 5 = maxdebug). Be aware that the library doesn't have the same notion of message importances as Messaging Server and that setting the level to *debug* can result in a lot of meaningless data. Also the `http_sso` log file is not managed by common Messaging Server logging code and is never cleaned up or rolled over. It is the responsibility of the system administrator to clean it up when setting the log level higher than the default. Default: 3
`local.webmail.sso.singlesignoff`	Single sign-off from Messaging Server to Access Manager. Access Manager is the central authentication authority, and single sign-off is always enabled from Access Manager to Messaging Server. This option allows a site to configure whether the *logout* button in webmail should also log the user out of Access Manager (saving some customization work). By default this is enabled. If this is disabled, a user logging out of the default webmail client is automatically logged back in since logout refers to the root document and the root document refers to the inbox display as long as the Access Manager cookie exists and is valid. Therefore, a site choosing to disable this option needs to customize what happens at webmail logout. Default: yes

Troubleshooting SSO

If there is a problem with SSO, the first thing to do is check the webmail log file *msg-svr-base*`/log/http` for errors. Increasing the logging level might be helpful (`configutil -o logfile.http.loglevel -v debug`). If this does not help, then check the `amsdk` messages in *msg-svr-base*`/log/http_sso`, then increase the `amsdk` logging level (`configutil -o local.webmail.sso.amloglevel -v 5`). New logging levels only take effect after server restart.

If SSO is still having problems, make sure you are using fully qualified host names of both Access Manager and Messaging Server during log in. Cookies are only shared between servers of the same domain, and browsers do not know what the domain is for local server names, so one must use the fully qualified names in the browser for SSO to work.

Trusted Circle SSO (Legacy)

This section describes trusted circle SSO. We do not recommend using this method of SSO since all future development will be with the Access Manager. However, there is some functionality available with trusted circle SSO that is not available with Access Manager SSO at this time. This section consists of the following sections:

Trusted Circle SSO Overview and Definitions

Before deploying SSO it is important to understand the following terminology.

- *SSO:* Single Sign-On. The ability to sign on to one application and be able access the other applications. The user identification is the same throughout all applications.
- *Trusted applications*. Applications sharing the SSO scheme (*SSO Prefix*) and trusting each other's cookies and verifications. Also known as *Peer SSO applications*.
- *Trusted circle*. The circle of trusted applications. They share the same SSO Prefix.
- *SSO Prefix*. A string defined by the person deploying SSO and made known to applications so they can use it to find cookies generated by other applications in the same trusted circle. Applications with different prefixes are not in the same circle and the user needs to re-authenticate when moving between these applications. The prefix sometimes, but not always, explicitly contains the trailing - ("-") in the configuration setting.
- *Application ID*. (appid). A unique string defined by the person deploying SSO for each application in the SSO circle.
- *SSO Cookie*. A token that the browser uses to remember that the user has authenticated to some application. The name of the cookie is of the form *SSO_prefix-application ID*. The value of the cookie is the SSO key, usually a session ID generated by the application.
- *Cookie Domain*. A domain within which the application is restricted to send cookies. This is a domain in the DNS sense.
- *Verification URL*. A URL used by one application to verify the cookie it found to another application.

Trusted Circle SSO Applications

Before implementing SSO, you must first consider which applications will be in this trusted circle. The applications which can be in this trusted circle are Messenger Express, Calendar Express, and the old iPlanet Delegated Administrator for Messaging (not recommended because it only supports Oracle LDAP Schema 1).

The following table shows which applications can be accessed from each other via SSO. From a user's point of view, logging into one of the applications on the first column, SSO works if the user is able to access application across the top row without having to re-enter user id and passwords.

SSO Interoperability

To:From:	Calendar Express	Messenger Express	Delegated Administrator
Calendar Express	SSO	SSO	SSO
Messenger Express	SSO	N/A	SSO
Delegated Administrator	SSO	SSO	N/A

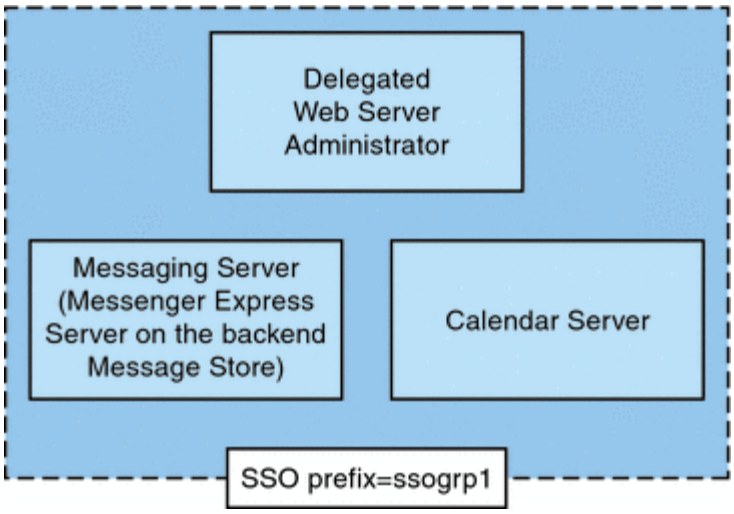
Trusted Circle SSO Limitations

- SSO Applications working together must be in the same domain.
- SSO Applications must have access to each other's SSO verification URL.
- Browsers must support cookies.
- For security purposes, SSO should not be used on machines where the browser is run.
- To switch to a different identity, the browser needs to be restarted.
- Assuming single sign-off is enabled in both Messenger Express and for Calendar Server, if you log out of Calendar Server, then you are supposed to have to login again to Messenger Express. If you log out of Messenger Express, then you would have to login again into Calendar Server. However, it currently does not work this way. You may stay logged into one after logging out of another.

Example Trusted Circle SSO Deployment Scenarios

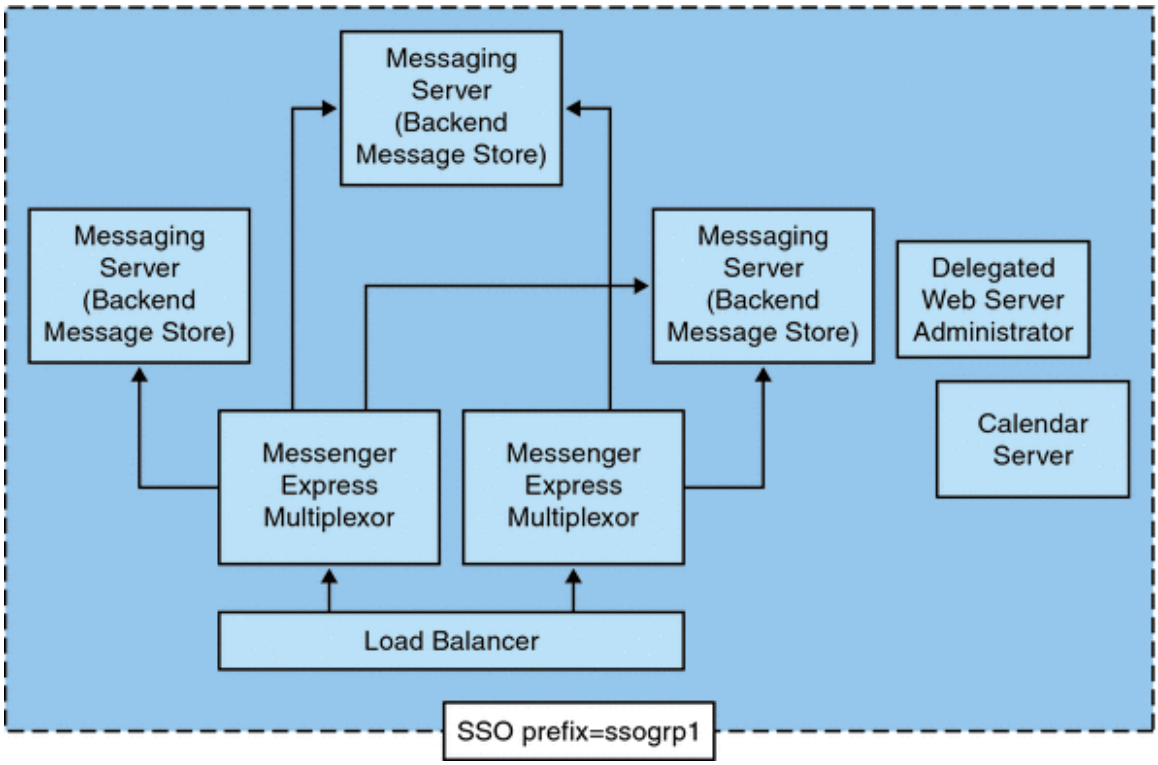
The simplest SSO deployment scenario consists of only Messenger Express and Delegated Administrator. A more complicated scenario can be created by adding Calendar Express, on the same or different machine, by using the same SSO prefix so they are in the same trusted circle. This is shown in the following figure.

Simple SSO Deployment



An even more complex deployment would include Webmail Servers and load balancers, as shown in the following figure.

Complex SSO Deployment



Setting Up Trusted Circle SSO

This section describes setting up SSO for Messenger Express, Delegated Administrator, and Calendar Server.

To Set Up SSO for Messenger Express, Delegated Administrator, and Calendar Server

1. Configure Messenger Express for SSO.
    a. Set the appropriate SSO `configutil` parameters.
    To enable single sign-on for Messenger Express with Delegated Administrator, set the configuration parameters as follows (assumes your default domain is `siroe.com`). These parameters are described in the Trusted Circle Single Sign-On Parameters table. You must logged in as `root` and change to the *instance_root* directory.

```
configutil -o local.webmail.sso.enable -v 1
configutil -o local.webmail.sso.prefix -v ssogrp1
```

    `ssogrp1` is the default SSO Prefix used by Delegated Administrator, although you can choose a different prefix, using the default would save a little typing when configuring Delegated Administrator and Calendar Server.

```
configutil -o local.webmail.sso.id -v ims5
```

    ims5 is a name you pick to identify Messenger Express (ME) to other applications.

```
configutil -o local.webmail.sso.cookiedomain -v ".siroe.com"
```

The above domain must match the domain used by the ME/browser client to connect to the servers. Thus, although the hosted domain on this server may be called `xyz.com`, use a real domain in the DNS. This value must start with a period.

```
configutil -o local.webmail.sso.singlesignoff -v 1
configutil -o local.sso.<ApplicationID>.verifyurl -v
"http://<ApplicationHost>:<port>/VerifySSO?"
```

ApplicationID is a name for the SSO application (example: ida for Delegated Administrator, ics50 for Calendar Server). *ApplicationHost*:*port* is the host and port number of the application. You will have one of these lines for each non-Messaging Server application. Example:

```
configutil -o local.sso.ida.verifyurl -v
"http://siroe.com:8080/VerifySSO?"
```

b. Restart Messenger Express http server after changing the configuration.

```
cd <instance_root>
./stop-msg http
./start-msg http
```

2. Configure Directory Server for SSO.
   a. Create a proxy user account in the directory.
   The proxy user account allows the Delegated Administrator to bind to the Directory Server for proxy authentication. Using the following LDIF code (`proxy.ldif`), you could create a proxy user account entry using `ldapadd`.
   `ldapadd -h mysystem.siroe.com -D "cn=Directory Manager" -w` *password* **`-v -f proxy.ldif`**

```
dn: uid=proxy, ou=people, o=siroe.com, o=isp
objectclass: top
objectclass: person
objectclass: organizationalperson
objectclass: inetorgperson
uid: proxy
givenname: Proxy
sn: Auth
cn: Proxy Auth
userpassword: proxypassword
```

   b. Create the appropriate ACIs for proxy user account authentication.
   Using the `ldapmodify` utility, create an ACI for each of the suffixes you created at the time you installed the Delegated Administrator.
   `osiroot` - The suffix you entered to store the user data (the default is `o=isp`). `osiroot` is the root of the Organization Tree.
   `dcroot` - The suffix you entered to store the domain information. (The default is `o=internet`.)
   `osiroot` - The suffix you entered to store the configuration information, it should have

been the same value you entered to store the user data.
The following is an example of an ACI entry (`aci1.ldif`) for the `osiroot` for the proxy user created earlier:

```
dn: o=isp
changetype: modify
add: aci
aci: (target="ldap:///o=isp")(targetattr="*")(version 3.0; acl
"proxy";allow (proxy) userdn="ldap:///uid=proxy, ou=people,
o=siroe.com, o=isp";)
```

```
ldapmodify -h siroe.com -D "cn=Directory Manager" -w password -v
-f aci1.ldif
```

Create a similar ACI entry (`aci2.ldif`) for the `dcroot`:

```
dn: o=internet
changetype: modify
add: aci
aci: (target="ldap:///o=internet")(targetattr="*")(version
3.0; acl
"proxy";allow (proxy) userdn="ldap:///uid=proxy, ou=people,
o=siroe.com, o=isp";)
```

```
ldapmodify -h siroe.com -D "cn=Directory Manager" -w password -v
-f aci2.ldif
```

3. Configure the Delegated Administrator
    a. Add the proxy user credentials and cookie name for context to the Delegated Administrator `resource.properties` file.
    Uncomment and modify the following entries in the Delegated Administrator `resource.properties` file:

```
LDAPDatabaseInterface-ldapauthdn=_Proxy_Auth_DN_
LDAPDatabaseInterface-ldapauthpw=_Proxy_Auth_Password_
NDAAuth-singleSignOnId=_SSO_Prefix-_
NDAAuth-applicationId=_DelAdminID_
```

For example:

```
LDAPDatabaseInterface-ldapauthdn=
uid=proxy,ou=people,o=cesta.com,o=isp
LDAPDatabaseInterface-ldapauthpw=proxypassword
NDAAuth-singleSignOnId=ssogrp1-
NDAAuth-applicationId=ida
```

The `resource.properties` file is stored in the following location:
iDA_svr_base`/nda/classes/netscape/nda/servlet/`

b. Add the participating server's verification URL.
To verify a single sign-on cookie it receives, Delegated Administrator must know who to contact. You must provide a verification URL for all known participating servers. Following the example, assume Messenger Express is installed and its application ID is `msg5`. Edit the Delegated Administrator `resource.properties` file and add an entry such as:

```
verificationurl-ssogrp1-msg5=http://<webmail_hostname>:<port>/Verify
```

The `resource.properties` file is located in the following directory:
iDA_svr_base`/nda/classes/netscape/nda/servlet/`

4. Add Delegated Administrator single sign-on cookie information and enable UTF8 Parameter Encoding.
   a. Define a context identifier for Delegated Administrator.
   Edit the `servlets.properties` file and uncomment all lines containing the text `servlet.`**`.context=ims50.`** **Where** is any string.
   The `servlets.properties` file is located at:
   Web_Svr_Base`/https-`*instancename*`/config/`
   b. Specify a cookie name for the context in the Enterprise Server configuration.
   Edit the Enterprise Server `contexts.properties` file and add the following line to the bottom of the file before the `#IDACONF-Start` line:
   `context.ims50.sessionCookie=ssogrp1-ida`
   The `contexts.properties` file is located at:
   Web_Svr_Base`/https-`*instancename*`/config/`
   c. Enable UTF8 parameter encoding for `ims5` contexts.
   To Enable UTF8 Parameter Encoding for `ims5` Contexts in the Enterprise Server configuration add the following entry to the Enterprise Server `contexts.properties` file:
   `context.ims50.parameterEncoding=utf8`
5. Restart Messenger Express.
   After you've made the configuration changes described in steps 1a through 2c, you must restart Messenger Express for the changes to take effect:

```
<Web_Svr_Base>/https-<instance_name>/stop
<Web_Svr_Base>/https-<instancename>/start
```

6. If you are deploying Calendar in this SSO group, configure Calendar Server.
   Edit `ics.conf` and add the following:

```
sso.appid = "ics50"
sso.appprefix = "ssogrp1"
sso.cookiedomain = ".red.iplanet.com"
sso.enable = "1"
sso.singlesignoff = "true"
sso.userdomain = "mysystem.red.iplanet.com"
sso.ims5.url="http://mysystem.red.iplanet.com:80/VerifySSO?"
sso.ida.url=http://mysystem.red.iplanet.com:8080/VerifySSO?
```

7. Restart Calendar Server
   `start-cal`
8. Restart the Messenger Express http server:

```
<msg-svr-base>/sbin/stop-msg http
<msg-svr-base>/sbin/start-msg http
```

Messenger Express Trusted SSO Configuration Parameters

You can modify the single sign-on configuration parameters for Messenger Express, shown in Messenger Express Trusted SSO Configuration Parameters, by using the `configutil` command. For more information about `configutil`, see *Messaging Server Administration Reference*.

Trusted Circle Single Sign-On Parameters

Parameter	Description
local.sso.*appid*.verifyurl	Sets the verify URL values for peer SSO applications. *appid* is the application ID of a peer SSO application whose SSO cookies are to be honored. For example, the default *appid* for Delegated Administrator is nda45.Its actual value is specified by the Delegated Administrator resource.properties file entry NDAAuth-applicationID. There should be one parameter defined for each trusted peer SSO application. The standard form of the verify URL is: http://*nda-host*:*port*/VerifySSO? If you are using a load balancer in front of multiple Webmail Servers and Message Store servers (running Messenger Express) or Calendar front ends, be sure to assign a *different* appid for each physical system with the real host names in the verifyurl. This will ensure that the correct system will be used to verify the cookie.
local.webmail.sso.cookiedomain	The string value of this parameter is used to set the cookie domain value of all SSO cookies set by the Messenger Express HTTP server. The default value is null. This domain must match the DNS domain used by the Messenger Express browser to access the server. It is not the hosted domain name.
local.webmail.sso.enable	Enables or disables all single sign-on functionality, including accepting and verifying SSO cookies presented by the client when the login page is fetched, returning an SSO cookie to the client on successful login and responding to requests from other SSO partners to verify its own cookies. If set to any non-zero value, the server performs all SSO functions. If set to zero, the server does not perform any of these SSO functions.The default value is zero.
local.webmail.sso.id	The string value of this parameter is used as the application ID value when formatting SSO cookies set by the Messenger Express HTTP server. The default value is null. This is an arbitrary string. Its value must match what you specify for the Delegated Administrator in its resource.properties file. The corresponding entry in resource.properties would be: Verifycationurl-XXX-YYY=http://webmailhost:webmailport/VerifySSO?Where XXX is the local.webmail.sso.prefix value set above, and YYY is the value of local.webmail.sso.id set here.
local.webmail.sso.prefix	The string value of this parameter is used as the prefix value when formatting SSO cookies set by the Messenger Express HTTP server. Only SSO cookies with this prefix will be recognized by the server; all other SSO cookies will be ignored. A null value for this parameter effectively disables all SSO functionality on the server. The default value is null. This string must match what is used by the Delegated Administrator in its resource.properties file without the trailing -. For example, if: NDAAuth-singleSignOnID=ssogrp1- Then this value should be set here to ssogrp1.
local.webmail.sso.singlesignoff	The integer value of this parameter, if set to any non-zero value, clears all SSO cookies on the client with prefix values matching the value configured in local.webmail.sso.prefix when the client logs out. If set to zero, Messenger Express will clear its own SSO cookie when the client logs out. The default value is zero.

Chapter 6. MTA Concepts

MTA Concepts

This information provides a conceptual description of the Messaging Server MTA.

Topics:

The MTA Functionality

The Message Transfer Agent, or *MTA* is a component of the Messaging Server. At its most basic level, the MTA is a message router. It accepts messages from other servers, reads the address, and routes it to the next server on way to its final destination, typically a user's mailbox.

Over the years, a lot of functionality has been added to the MTA, and with it, size, power, and complexity. These MTA functions overlap, but, in general, can be classified as follows:

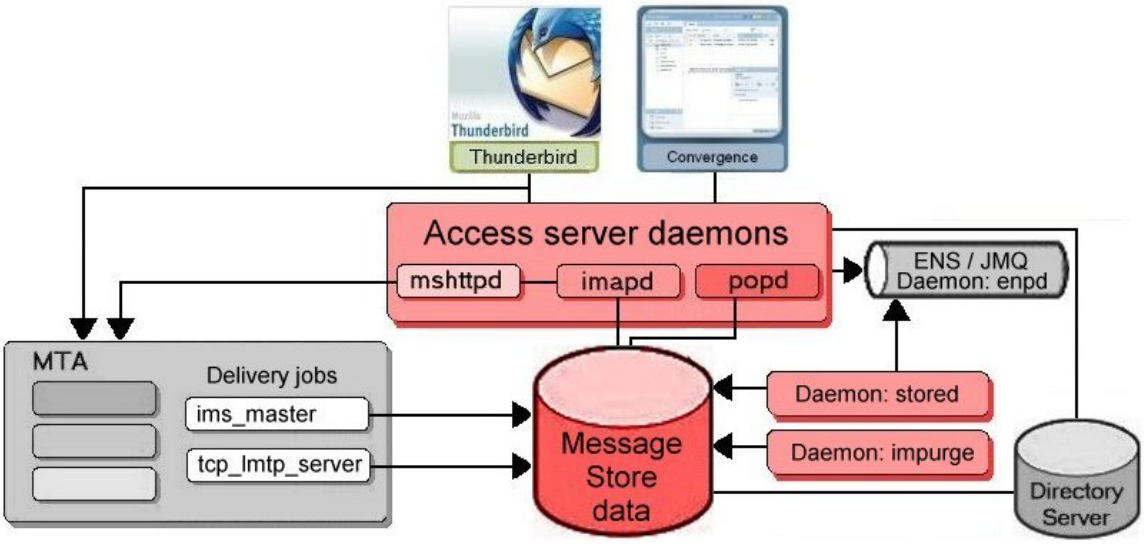
- **Routing.** Accepts a message, expands or transforms it if necessary (for example if it is an alias), and routes it to the next server, channel, program, file, or whatever. The routing function has been expanded to allow administrator specification of the internal and external mechanics of how messages are routed. For example, it is possible to specify things such as SMTP authentication, use of various SMTP commands and protocol, TCP/IP or DNS lookup support, job submission, process control and message queueing and so on.
- **Address Rewriting.** Envelope addresses are often rewritten as part of the routing process, but envelope or header addresses can also be rewritten to a more desired or appropriate form.
- **Filtering.** The MTA can filter messages based on address, domain, possible virus or spam content, size, IP address, header content, and so on. Filtered messages can be discarded, rejected, modified, sent to a file, sent to a program, or be sent to the next server on its way to a user mailbox.
- **Content Modification.** Message headers or content can be modified. Example: making a message readable to a specific client or in a specific character set or checking for spam or viruses.
- **Auditing.** Tracking who submitted what, where and when.

A number of subcomponents and processes support these functions and are shown in the MTA Architecture figure. This information describes these subcomponents and processes. In addition, a number of tools allow system administrators to enable and configure these functions. These include MTA options, `configutil` parameters, mapping tables, keywords, channels, and rewrite rules. These are described in the following MTA information:

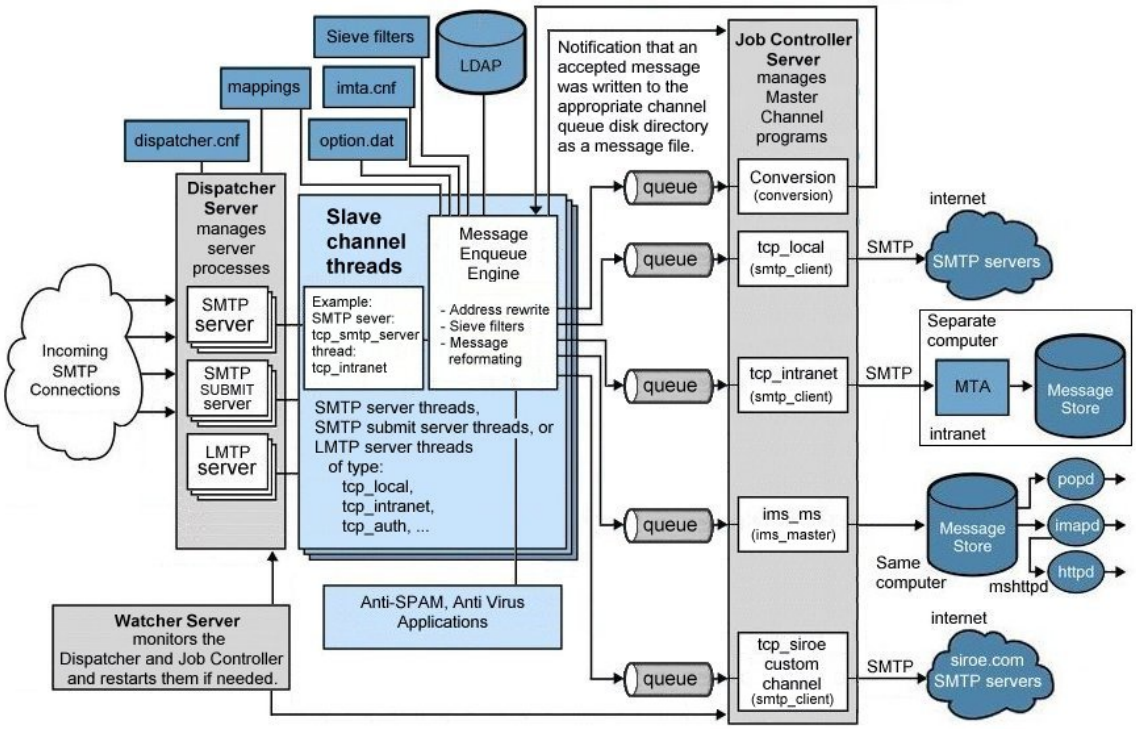
- About MTA Services and Configuration
- Configuring Rewrite Rules
- Configuring Channel Definitions
- Using Predefined Channels
- Integrating Spam and Virus Filtering Programs Into Messaging Server

- LMTP Delivery
- Vacation Automatic Message Reply
- Mail Filtering and Access Control
- Security and Access Control
- Managing Logging
- Troubleshooting the MTA
- Monitoring Messaging Server

High Level Message Store and MTA Architecture



MTA Architecture



MTA Architecture and Message Flow Overview

This section provides a short overview of MTA architecture and message flow (see the MTA Architecture figure). The MTA is a highly complex component and this figure is a *simplified* depiction of messages flowing through the system. In fact, this picture is not a perfectly accurate depiction of all messages flowing through the system. For purposes of conceptual discussion, however, it must suffice.

Dispatcher and SMTP Server (Slave Program)

Messages enter the MTA from the Internet or intranet via SMTP sessions. When the MTA receives a request for an SMTP connection, the MTA *dispatcher* (a multithreaded connection dispatching agent), executes a *slave* program (`tcp_smtp_server`) to handle the SMTP session. The dispatcher maintains pools of multithreaded processes for each service. As additional sessions are requested, the dispatcher activates an SMTP server program to handle each session. A process in the Dispatcher's process pool may concurrently handle many connections. Together the dispatcher and slave program perform a number of different functions on each incoming message. Three primary functions are:

- **Message blocking.** Messages from specified IP addresses, mail addresses, ports, channels, header strings and so on, may be blocked (see Mail Filtering and Access Control).
- **Address changing.** Incoming `From:` or `To:` addresses may be rewritten to a different form.
- **Channel enqueueing.** Addresses are run through the rewrite rules to determine which channel the message should be sent.

For more information, see The Dispatcher.

Routing and Address Rewriting

SMTP servers enqueue messages, but so can a number of other channels including, the conversion channel and reprocess channel. A number of tasks are achieved during this phase of delivery, but the primary tasks are:

- Alias expansion.
- Running the addresses through the rewrite rules which do two things:
  - Rewrite the domain part of addresses into a desired format.
  - Direct messages to the appropriate channel queue.

Channels

The channel is the fundamental MTA component used for message processing. A channel represents a message connection with another system (for example, another MTA, another channel, or the local message store). As mail comes in, different messages require different routing and processing depending on the message's source and destination. For example, mail to be delivered to a local message store will be processed differently from mail to be delivered to the Internet, which will be processed differently from mail to be sent to another MTA within the mail system. Channels provide the mechanism for customizing the processing and routing required for each connection. In a default installation, the majority of messages go to a channels handling Internet, intranet, and local messages.

Specialized channels for specific situations can also be created. For example, suppose that a certain Internet domain processes mail very slowly causing mail addressed to this domain to clog up the MTA. A special channel could be created to provide special handling for messages addressed to the slow domain, thus relieving the system of this domain bottleneck.

The domain part of the address determines to what channel the message is enqueued. The mechanism for reading the domain and determining the appropriate channel is called the rewrite rules (see Rewrite Rules).

Channels typically consist of a channel queue and a channel processing program called a *master program*. After the slave program delivers the message to the appropriate channel queue, the master program performs the desired processing and routing. Channels, like rewrite rules, are specified and configured in the `imta.cnf` file. Here is an example of a channel entry:

```
tcp_intranet smtp mx single_sys subdirs 20 noreverse maxjobs 7 SMTP_POOL
maytlsserver allowswitchchannel saslswitchchannel tcp_auth
tcp_intranet-daemon
```

The first word, in this case `tcp_intranet` is the channel name. The last word is called the channel tag. The words in between are called channel options (formerly called channel keywords) and specify how messages are to be processed. Hundreds of different options enable messages to be processed in many ways. A complete description of channel options is provided in Configuring Channel Definitions.

Message Delivery

After the message is processed, the master program sends the message to the next stop along the message's delivery path. This may be the intended recipient's mailbox, another MTA, or even a different channel. Forwarding to another channel is not shown in the picture, but is a common occurrence.

The local parts of addresses and received fields are typically 7-bit characters. If the MTA reads 8-bit characters in the in these fields, it replaces each 8-bit character with asterisks.

The Dispatcher

The Dispatcher is a multithreaded dispatching agent that permits multiple multithreaded server processes to share responsibility for SMTP connection services. When using the Dispatcher, it is possible to have several multithreaded SMTP server processes running concurrently, all handling connections to the same port. In addition, each server may have one or more active connections.

The Dispatcher acts as a central receiver for the TCP ports listed in its configuration. For each defined service, the Dispatcher may create one or more SMTP server processes to handle the connections after they are established.

In general, when the Dispatcher receives a connection for a defined TCP port, it checks its pool of available worker processes for the service on that port and chooses the best candidate for the new connection. If no suitable candidate is available and the configuration permits it, the Dispatcher may create a new worker process to handle this and subsequent connections. The Dispatcher may also create a new worker process in expectation of future incoming connections. There are several configuration options which may be used to tune the Dispatcher's control of its various services, and in particular, to control the number of worker processes and the number of connections each worker process handles.

See Dispatcher Configuration File for more information.

Creation and Expiration of Server Processes

Automatic housekeeping facilities within the Dispatcher control the creation of new and expiration of old or idle server processes. The basic options that control the Dispatcher's behavior are `MIN_PROCS` and `MAX_PROCS`. `MIN_PROCS` provides a guaranteed level of service by having a number of server processes ready and waiting for incoming connections. `MAX_PROCS`, on the other hand, sets an upper limit on how many server processes may be concurrently active for the given service.

It is possible that a currently running 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 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 server processes. `MIN_CONNS` specifies the number of connections that flags a server

process as "busy enough" while MAX_CONNS specifies the "busiest" that a server process can be.

In general, the Dispatcher creates a new server process when the current number of server processes is less than MIN_PROCS or when all existing server processes are "busy enough" (the number of currently active connections each has is at least MIN_CONNS).

If a server process is killed unexpectedly, for example, by the UNIX system kill command, the Dispatcher still creates new server processes as new connections come in.

For information about configuring the Dispatcher, see Dispatcher Configuration File.

To Start and Stop the Dispatcher

To start the Dispatcher, execute the command:

```
start-msg dispatcher
```

This command subsumes and makes obsolete any other start-msg command that was used previously to start up a component of the MTA that the Dispatcher has been configured to manage. Specifically, you should no longer use imsimta start smtp. An attempt to execute any of the obsoleted commands causes the MTA to issue a warning.

To shut down the Dispatcher, execute the command:

```
stop-msg dispatcher
```

What happens with the server processes when the Dispatcher is shut down depends upon the underlying TCP/IP package. If you modify your MTA configuration or options that apply to the Dispatcher, you must restart the Dispatcher so that the new configuration or options take effect.

To restart the Dispatcher, execute the command:

```
imsimta restart dispatcher
```

Restarting the Dispatcher has the effect of shutting down the currently running Dispatcher, then immediately starting a new one.

Rewrite Rules

Rewrite rules determine the following:

- How to rewrite the domain part of an address into its proper or desired format.
- To which channel the message should be enqueued after the address is rewritten.

Each rewrite rule consists of a *pattern* and a *template*. The pattern is a string to match against the domain part of an address. The template specifies the actions to take if the domain part matches the pattern. It consists of two things:

1. A set of instructions (that is, a string of control characters) specifying how the address should be rewritten.
2. The name of the channel to which the message shall be sent. After the address is rewritten, the

message is enqueued to the destination channel for delivery to the intended recipient.

Here is an example of a rewrite rule:

```
siroe.com $U%$D@tcp_siroe-daemon
```

`siroe.com` is the domain pattern. Any message with the address containing `siroe.com` will be rewritten as per the template instructions (`$U%$D`). `$U` specifies that the rewritten address use the same user name. % specifies that the rewritten address use the same domain separator. `$D` specifies that the rewritten address use the same domain name that was matched in the pattern. `@tcp_siroe-daemon` specifies that the message with its rewritten address be sent to the channel called `tcp_siroe-daemon`. See Configuring Rewrite Rules for more details.

For more information about configuring rewrite rules, see The MTA Configuration File and Configuring Rewrite Rules.

Channels

The channel is the fundamental MTA component that processes a message. A channel represents a connection with another computer system or group of systems. The actual hardware connection or software transport or both may vary widely from one channel to the next.

Channels perform the following functions:

- Transmit messages to remote systems, deleting them from their queue after they are sent.
- Accept messages from remote systems, placing them in the appropriate channel queues.
- Deliver messages to the local message store.
- Deliver messages to programs for special processing.

Messages are enqueued by channels on the way into the MTA and dequeued on the way out. Typically, a message enters via one channel and leaves by another. A channel might dequeue a message, process the message, or enqueue the message to another MTA channel.

This section consists of the following subsections:

Master and Slave Programs

Generally (but not always), a channel is associated with two programs: master and slave. The slave program accepts messages from another system and adds them to a channel's message queue. The master program transfers messages from the channel to another system.

For example, an SMTP channel has a master program that transmits messages and a slave program that receives messages. These are, respectively, the SMTP client and server.

The master channel program is typically responsible for outgoing connections where the MTA has initiated the operation. The master channel program:

- Runs in response to a local request for processing.
- Dequeues the message from the channel message queue.
- If the destination format is not the same format as the queued message, performs conversion of addresses, headers, and content, as necessary.
- Initiates network transport of the message.

The slave channel program typically accepts incoming connections where the MTA is responding to an external request. The slave channel program:

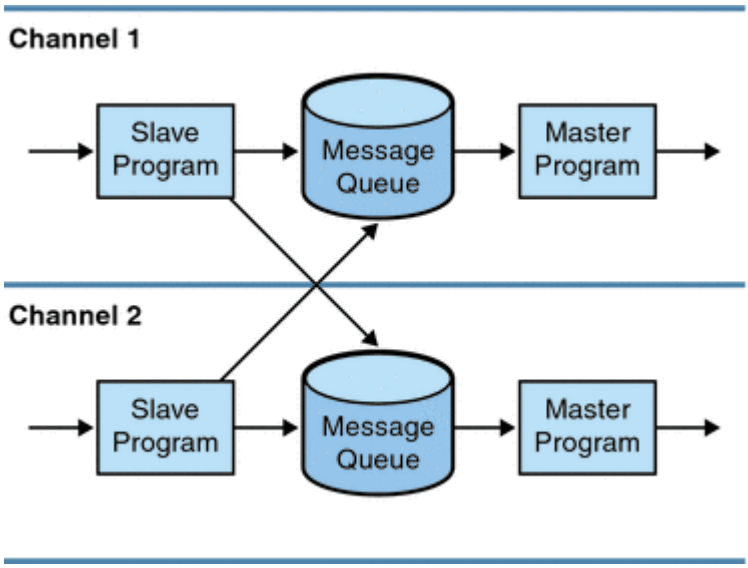
- Runs in response to an external event or upon local demand.
- Enqueues a message to a channel. The target channel is determined by passing envelope

addresses through a rewrite rule.

For example, Master and Slave Programs shows two channel programs, Channel 1 and Channel 2. The slave program in Channel 1 receives a message from a remote system. It looks at the address, applies rewrite rules as necessary, then based on the rewritten address enqueues the message to the appropriate channel message queue.

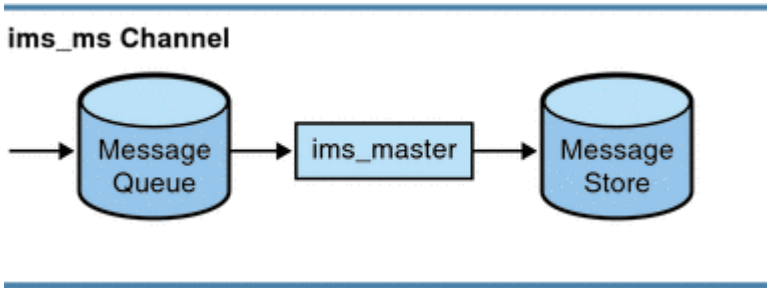
The master program dequeues the message from the queue and initiates network transport of the message. Note that the master program can only dequeue messages from its own channel queue.

Master and Slave Programs



Although a typical channel has both a master and a slave program, it is possible for a channel to contain only a slave program *or* a master program. For example, the `ims-ms` channel supplied with Messaging Server contains only a master program because this channel is responsible only for dequeuing messages to the local message store, as shown in ims-ms Channel.

ims-ms Channel



Channel Message Queues

All channels have an associated message queue. When a message enters the messaging system, a slave program determines to which message queue the message is enqueued. The enqueued messages are stored in message files in the channel queue directories. By default, these directories are stored at the following location: *msg-svr-base*`/data/queue/`*channel*`/*`. See the topic on message queue sizing in *Unified Communications Suite Deployment Planning Guide* for more information.

Caution
Do not add any files or directories in the MTA queue directory (that is, the value of `IMTA_QUEUE` in the `imta_tailor` file) as this will cause problems. When using a separate file system for the MTA queue directories, create a subdirectory under that mount point and specify that subdirectory as the value of `IMTA_QUEUE`. Beginning with Messaging Server 7, the `IMTA_QUEUE` tailor file option is no longer honored.

Channel Definitions

Channel definitions appear in the lower half of the MTA configuration file, `imta.cnf`, following the rewrite rules (see The MTA Configuration File rules section and the start of the channel definitions).

A channel definition contains the name of the channel followed by an optional list of keywords that define the configuration of the channel, and a unique channel tag, which is used in rewrite rules to route messages to the channel. Channel definitions are separated by single blank lines. Comments, but no blank lines, may appear inside a channel definition.

```
[blank line]
! sample channel definition
<Channel_Name> <keyword1> <keyword2>
<Channel_Tag>
[blank line]
```

Collectively, the channel definitions are referred to as the channel host table. An individual channel definition is called a channel block. In the following example, the channel host table contains three channel definitions or blocks.

```
! test.cnf - An example configuration file.
!
! Rewrite Rules
      .
      .
      .

! BEGIN CHANNEL DEFINITIONS
! FIRST CHANNEL BLOCK
l
local-host

! SECOND CHANNEL BLOCK
a_channel defragment charset7 usascii
a-daemon

! THIRD CHANNEL BLOCK
b_channel noreverse notices 1 2 3
b-daemon
```

A typical channel entry will look something like this:

```
tcp_intranet smtp mx single_sys subdirs 20 noreverse maxjobs 7 SMTP_POOL
maytlsserver allowswitchchannel saslswitchchannel tcp_auth
tcp_intranet-daemon
```

The first word, in this case `tcp_intranet`, is the channel name. The last word, in this case `tcp_intranet-daemon`, is called the *channel tag*. The channel tag is the name used by rewrite rules to direct messages. The words in between the channel name and channel tag are called channel options (formerly channel keywords) and specify how the message is to be processed. Hundreds of different keywords allow messages to processed in many ways. A complete listing of channel keywords is listed and described in Configuring Channel Definitions.

The channel host table defines the channels Messaging Server can use and the names of the systems associated with each channel.

On UNIX systems, the first channel block in the file always describes the local channel, `l`. (An exception is a `defaults` channel, which can appear before the local channel.) The local channel is used to make routing decisions and for sending mail sent by UNIX mail tools.

You can also set global options for channels in the MTA Option file, `option.dat`, or set options for a specific channel in a channel option file. For more information on the option files, see Option File, and TCP/IP (SMTP) Channel Option Files. For details on configuring channels, see Configuring Channel Definitions. For more information about creating MTA channels, see The MTA Configuration File.

The MTA Directory Information

For each message that it processes, the MTA needs to access directory information about the users, groups, and domains that it supports. This information is stored in an LDAP directory service. The MTA directly accesses the LDAP directory. This is fully described in MTA Address Translation and Routing.

The Job Controller

Each time a message is enqueued to a channel, the Job Controller ensures that there is a job running to deliver the message. This might involve starting a new job process, adding a thread, or simply noting that a job is already running. If a job cannot be started because the job limit for the channel or pool has been reached, the Job Controller waits until another job has exited. When the job limit is no longer exceeded, the Job Controller starts another job.

Channel jobs run inside processing pools within the Job Controller. A pool can be thought of a "place" where the channel jobs are run. The pool provides a computing area where a set of jobs can operate without vying for resources with jobs outside of the pool. For more information on pools see Job Controller File and Channel configuration.

Job limits for the channel are determined by the `maxjobs` channel keyword. Job limits for the pool are determined by the `JOB_LIMIT` option for the pool.

Messaging Server normally attempts to deliver all messages immediately. If a message cannot be delivered on the first attempt, however, the message is delayed for a period of time determined by the appropriate `backoff` keyword. As soon as the time specified in the `backoff` keyword has elapsed, the delayed message is available for delivery, and if necessary, a channel job is started to process the message.

The Job Controller's in-memory data structure of messages currently being processed and awaiting processing typically reflects the full set of message files stored on disk in the MTA queue area. However, if a backlog of message files on disk builds up enough to exceed the Job Controller's in-memory data

structure size limit, then the Job Controller tracks in memory only a subset of the total number of messages files on disk. The Job Controller processes only those messages it is tracking in memory. After a sufficient number of messages have been delivered to free enough in-memory storage, the Job Controller automatically refreshes its in-memory store by scanning the MTA queue area to update its list of messages. The Job Controller then begins processing the additional message files it just retrieved from disk. The Job Controller performs these scans of the MTA queue area automatically.

Prior to Messaging Server 6.2, the Job Controller read all the files in the queue directory in the order in which they are found. Starting with Messaging Server 6.2, the Job Controller now reads several channel queues at the same time. This makes for much more reasonable behavior on startup, restart, and after `max_cache_messages` (formerly `max_messages`) has been exceeded. The number of channel queues to be read at once is controlled by the Job Controller option `Rebuild_Parallel_Channels`. This can take any value between 1 and 100. The default is 12.

If your site routinely experiences heavy message backlogs, you might want to tune the Job Controller by using the `MAX_MESSAGES` option. By increasing the `MAX_MESSAGES` option value to allow Job Controller to use more memory, you can reduce the number of occasions when message backlogs overflow the Job Controller's in-memory cache. The Job controller in-memory cache contains envelope destination and queuing/retry schedule information, not the entire message. The in-memory cache size is approximately 512 bytes per message. This reduces the overhead involved when the Job Controller must scan the MTA queue directory. Keep in mind, however, that when the Job Controller does need to rebuild the in-memory cache, the process will take longer because the cache is larger. Note also that because the Job Controller must scan the MTA queue directory every time it is started or restarted, large message backlogs mean that starts or restarts of the Job Controller will incur more overhead than starts or restarts when no such backlog exists.

You do not want to overwhelm the job controller by keeping information about huge numbers of messages in memory. For this reason, there has to be a and upper and lower limit. The number specified by `MAX_MESSAGES` is the number of messages that the job controller will hold in memory. It will get this high if there are new messages delivered, for instance ones received by tcp_smtp_server. Beyond this number, messages are queued (put on disk), but not put into the job controller memory structure. The job controller notices this condition and when the number of messages in memory drops below half this maximum, it starts scanning the disk queues for more messages. It always looks for untried messages "ZZ..." files first, then previously tried messages.

In addition, the job controller limits the number of messages reclaimed from disk. It only reads from disk up to three-quarters of the `MAX_MESSAGES` to allow for headroom for new messages (if messages are being reclaimed from disk, they have been delayed, which is an undesirable state).

Furthermore, you want to avoid cluttering up the memory structure with delayed messages (those that cannot be processed yet). When a message is delayed because it cannot be delivered immediately (a delivery attempt has failed if the number of messages the job controller knows about is greater than 5/8 of `MAX_MESSAGES` and the number of delayed messages is greater than 3/8 of `MAX_MESSAGES`) the message is forgotten until the next sweep of the on disk structures, which will be when the number of messages drops below 1/2 `MAX_MESSAGES`.

The only obvious problems with having `MAX_MESSAGES` too small is that the scheduling of jobs will become suboptimal. The scanning of the disk queues is also a bit simplistic. If you have huge numbers of messages backlogged in both the tcp_local and ims_ms queues, then the rebuild thread will find all the messages for one channel first, then the ones for the next channel. This can result in alarmed administrators reporting that they've fixed one issue, but are only seeing only one specific channel dequeuing.

This is not a problem. There is a memory cost of approximately 140 bytes for each message. Having a message limit of 100000, you are limiting the job controller data structures to about 20 Megabytes (there are other data structures representing jobs, channels, destination hosts and so on). This is insignificant on a big server.

All the named objects in the job controller are tracked in a hash table. This is sized at the next power of 2

bigger than MAX_MESSAGES, and is never re-sized. Each entry in this hash table is a pointer, so we are looking at a memory usage of four times MAX_MESSAGES rounded up to a power of two. Being a hash table, this will tend all to be in memory as the hash function is supposed to be random. This is another 0.5 Megabytes in the default case.

For information about pools and configuring the Job Controller, see Job Controller File and see Channel configuration.

To Start and Stop the Job Controller

To start the Job Controller, execute the command:

```
start-msg job_controller
```

To shut down the Job Controller, execute the command:

```
stop-msg job_controller
```

To restart the Job Controller, execute the command:

```
imsimta restart job_controller
```

Restarting the Job Controller has the effect of shutting down the currently running Job Controller, then immediately starting a new one.

Chapter 7. MTA Address Translation and Routing

MTA Address Translation and Routing

Prior to Messaging Server 6 2003Q4, Messaging Server used to access all user, domain, and group data from a database that was compiled from information stored in an LDAP server. When directory information was updated in the LDAP server, the database information was synchronized with a program called `dirsync`. The Messaging Server MTA now accesses the LDAP directory directly. This information describes the flow of data through the MTA using direct LDAP data access.

Topics:

The Direct LDAP Algorithm and Implementation

The following sections describe direct LDAP processing:

Domain Locality Determination

Starting with an address of the form *user@domain*, the address translation and routing process first checks to see if *domain* is local.

This section consists of the following subsections:

Rewrite Rule Machinery

A facility has been added to the MTA rewrite rule machinery to check a given string to see if it is a domain that needs to be locally handled. This new facility is activated by a `$V` or `$Z` metacharacter. These new metacharacters are syntactically similar to the existing `$N`, `$M`, `$Q`, and `$C` metacharacters, that is, they are followed by a pattern string. In the case of `$N`, `$M`, `$Q`, and `$C` the pattern is matched against either the source or destination channel. In the case of `$V` and `$Z` the pattern is a domain and the check is to see if it is local. `$V` causes a rule failure for a nonlocal domain and `$Z` causes a rule failure for a local domain.

The handling of these metacharacters is implemented as the following procedure:

1. Messaging Server checks to see if the current domain matches a valid domain entry in the directory. Go to step 3 if no entry exists.
2. If the domain has an entry in the directory, the attribute specified by the LDAP_DOMAIN_ATTR_ROUTING_HOSTS MTA option (default mailRoutingHosts) is retrieved from the domain entry. If this attribute is present, it lists the set of hosts able to handle users in this domain. This list is compared against the host specified by the local.hostname configutil parameter and the list of hosts specified by the local.imta.hostnamealiases configutil parameter. These options can be overridden by the LDAP_LOCAL_HOST and LDAP_HOST_ALIAS_LIST MTA options, respectively. If there is a match or the attribute is not present on the domain, the domain is local. If no match occurs, the domain is nonlocal.
   The handling of domains considered to be nonlocal because of the mailRoutingHosts attribute depends on the setting of the ROUTE_TO_ROUTING_HOST MTA option. If the option is set to 0 (the default), the address is simply treated as non-local and MTA rewrite rules are used to determine routing. If the option is set to 1, a source route consisting of the first value listed in the LDAP_DOMAIN_ATTR_ROUTING_HOSTS MTA option is prepended to the address.
3. If no domain entry can be found, remove a component from the left hand side of the domain and go to Step 1. If no components remain continue to Step 4.
   A consequence of this backtracking up the domain tree is that if siroe.com is recognized as local, any subdomain of siroe.com will be recognized as local. Situations might arise where this is undesirable, so an MTA option, DOMAIN_UPLEVEL, is provided to control this behavior. Specifically, bit 0 (value = 1) of DOMAIN_UPLEVEL, if clear, disables retries with domain components removed. The default value of DOMAIN_UPLEVEL is 0.
4. Vanity domain checking now needs to be performed. Vanity domains do not have domain entries, rather, they are specified by attaching special domain attributes to one or more user entries. The vanity domain check is done by instituting an LDAP search by using the LDAP URL specified by the DOMAIN_MATCH_URL MTA option. The value of this option should be set to:
   ldap:///$B?msgVanityDomain?sub?(msgVanityDomain=$D)
   $B substitutes the value of the local.ugldapbasedn configutil parameter. This is the base of the user tree in the directory. The LDAP_USER_ROOT MTA option can be used to override the value of this configutil option specifically for the MTA.
   The actual return value from this search does not matter. What matters is if there is a value to return. If there is a return value the domain is considered to be local, if not it is considered to be nonlocal.

Domain Map Determination of Domain Locality

The following steps are performed to find valid domain entries in the directory. These steps are schema-level specific. For Oracle LDAP Schema 1, the steps are the following:

1. Convert the domain to a base DN in the domain tree.
   This is done by converting the domain into a series of dc components and then adding a domain root suffix. The default suffix is obtained from the service.dcroot configutil parameter. The default suffix is o=internet. So a domain of the form a.b.c.d would typically be converted into dc=a,dc=b,dc=c,dc=d,o=internet. The service.dcroot configutil parameter can be overridden by setting the LDAP_DOMAIN_ROOT MTA option.
2. Look for an entry with the base DN found in Step 1 and an object class of either inetDomain or inetDomainAlias.
   The search filter used for this purpose can be overridden by setting the LDAP_DOMAIN_FILTER_SCHEMA1 MTA option which defaults to (|(objectclass=inetDomain)(objectclass=inetdomainalias)).
3. Exit with a failure if nothing is found.
4. If the object class of the entry found is inetDomain, check to make sure the entry has an inetDomainBaseDn attribute associated with the domain entry.
   If it is present, it is saved for use in subsequent searches for user entries and processing terminates. If it is not present, the entry is assumed to be a domain alias and processing continues with step 5. The MTA option LDAP_DOMAIN_ATTR_BASEDN can be used to override the use of inetDomainBaseDN.
5. The entry must be a domain alias. Look up the new entry referenced by the

aliasedObjectName attribute and return to step 4.
Processing terminates with a failure if the no aliasedObjectName attribute is present. An alternative to the use of aliasedObjectName attribute can be specified with the MTA option LDAP_DOMAIN_ATTR_ALIAS.
Processing can return to step 4 at most once. Domain aliases pointing at domain aliases are not allowed.

In Sun LDAP Schema 2, the action taken is much simpler: The directory is searched for an entry with the object class sunManagedOrganization where the domain appears as a value of either the sunPreferredDomain or associatedDomain attribute. If need be, the use of the sunPreferredDomain and associatedDomain attributes for this purpose can be overridden with the respective MTA options LDAP_ATTR_DOMAIN1_SCHEMA2 and LDAP_ATTR_DOMAIN2_SCHEMA2. The search is done under the root specified by the service.dcroot configutil parameter. The service.dcroot configutil parameter can be overridden by setting the LDAP_DOMAIN_ROOT MTA option. Additionally, domain entries in Schema 2 are not required to have inetDomainBaseDn attributes. If they do not, the base of the user tree is assumed to be the domain entry itself.

Two MTA options support more efficient domain lookups from user base domain names. They are LDAP_BASEDN_FILTER_SCHEMA1, which is a string specifying a filter used to identify Schema 1 domains when performing user base domain name searches. The default is the value of LDAP_DOMAIN_FILTER_SCHEMA1 if that MTA option is specified. If neither option is specified the default is (objectclass=inetDomain). LDAP_BASEDN_FILTER_SCHEMA2 is a string specifying additional filter elements used to identify Schema 2 domains when performing user base domain name searches. The default is the value of LDAP_DOMAIN_FILTER_SCHEMA2, if that MTA option is specified. If neither option is specified, the default is an empty string.

Caching Of Domain Locality Information

Due to the frequency with which domain rewrite operations are performed and the expense of the directory queries (especially the vanity domain check) both negative and positive indications about domains need to be cached. This is implemented with an in-memory open-chained dynamically-expanded hash table. The maximum size of the cache is set by the DOMAIN_MATCH_CACHE_SIZE MTA option (default 100000) and the timeout for entries in the cache is set by the DOMAIN_MATCH_CACHE_TIMEOUT MTA option (default 600 seconds).

Error Handling

Temporary server failures during this process have to be handled carefully, since when they occur, it is impossible to know whether or not a given domain is local. Basically two outcomes are possible in such a case:

1. Return a temporary (4_xx_) error to the client telling it to try the address again later.
2. Accept the address but queue it to the reprocessing channel so it can be retried locally later.

Neither of these options is appropriate in all cases. For example, outcome 1 is appropriate when talking to a remote SMTP relay. But outcome 2 is appropriate when dealing with an SMTP submission from a local user.

While it would be possible in theory to handle temporary failures by using multiple rules with the same pattern, the overhead of repeating such queries, even with a cache in place, is unacceptable. For these reasons, the simple success/fall-through-to-the-next-rule matching model of domain rewriting is inadequate. Instead, a special template, specified by the MTA option DOMAIN_FAILURE, is used in the event of a domain lookup failure. When a $V operation fails, this template replaces the remainder of the current rewrite rule template being processed.

Pattern for Domain Check Rewrite Rule

This domain check needs to be performed before other rewrite rules have a chance to operate. This

ordering is insured by using the special $* on the left hand side in the rule. The $* pattern is checked prior to any other rules.

Putting It All Together

Taking all the machinery described so far into account, the imta.cnf file needs the following rewrite rule:

```
$*      $E$F$U%$H$V$H@localhost
```

and the DOMAIN_FAILURE MTA option in the option.dat file needs the following value:

```
reprocess-daemon$Mtcp_local$1M$1~-error$4000000?Temporary lookup failure
```

In this rewrite rule, localhost is the host name associated with the local channel. The value of the DOMAIN_FAILURE option shown here is the default value so it does not need to appear in option.dat under normal circumstances.

The ordering here is especially tricky. The MTA checks $V after the address is rebuilt but before the route is added. This lets the MTA to change the route in the event of a temporary lookup failure. Pending channel match checks are applied any time the insertion point changes, so the @ after the second $H invokes the check. If the check succeeds, the remainder of the template applies and rewrite processing concludes. If the check fails, the rewrite fails and rewriting continues with the next applicable rewrite rule. If the check cannot be performed due to a temporary failure, template processing continues with the value specified by the DOMAIN_FAILURE MTA option. The value of this template first sets the routing host to reprocess-daemon. Then the template checks to see whether or not the MTA is dealing with a reprocessing channel of some sort or tcp_local. If the MTA is dealing with such a channel, the rule continues, making the routing host illegal and specifying a temporary failure as the outcome. If the MTA is not dealing with such a channel, the rule is truncated and successfully terminated, thereby rewriting the address to the reprocess channel.

Alias Expansion of Local Addresses

Once an address has been determined to be associated with the local channel it automatically undergoes alias expansion. The alias expansion process examines a number of sources of information, including:

1. The alias file (part of the compiled configuration)
2. The alias database
3. Alias URLs

The exact alias sources that are checked and the order in which they are checked depends on the setting of the ALIAS_MAGIC MTA option in the option.dat file. For direct LDAP, set the option to 8764. This means that the URL specified by the ALIAS_URL0 MTA option is checked first, then the URL specified by the ALIAS_URL1 MTA option, then the URL specific by the ALIAS_URL2 MTA option, and finally, the alias file. The alias database is not checked when this setting is active.

The following sections further describe alias expansion:

- Alias Checking with LDAP URLs
- The $V Metacharacter
- Calling a Mapping From a URL
- The $R Metacharacter
- Determining the Attributes to Fetch
- Handling LDAP Errors
- Sanity Checks on the LDAP Result

- Support for Vanity Domains
- Support for Catchall Addresses

Alias Checking with LDAP URLs

Checking of aliases in LDAP is implemented by specifying two special LDAP URLs as alias URLs. The first of these handles regular users and groups; vanity domains are handled by subsequent alias URLs. The first URL is specified as `ALIAS_URL0`:

```
ALIAS_URL0=ldap:///$V?*?sub?$R
```

The $V Metacharacter

Metacharacter expansion occurs prior to URL lookup. The two metacharacters used in the `ALIAS_URL0` value are `$V` and `$R`.

The $V metacharacter converts the domain part of the address into a base DN. This is similar to the initial steps performed by the $V rewrite rule metacharacter described previously in the section entitled Rewrite Rule Machinery. $V processing consists of the following steps:

1. Get the base DN for user entries in the current domain.
2. Get the canonical domain associated with the current domain. In Oracle LDAP Schema 1, the canonical domain name is given by the `inetCanonicalDomainName` attribute of the domain entry if the attribute is present. If the attribute is absent the canonical name is the name constructed in the obvious way from the DN of the actual domain entry. This will differ from the current domain when the current domain is an alias. The name attribute used to store the canonical name may be overridden with the `LDAP_DOMAIN_ATTR_CANONICAL` MTA option in the `option.dat` file.
   In Sun LDAP Schema 2, the canonical name is simply the value of the `SunPreferredDomain` attribute.
   A utility for verifying canonical domain settings for domains with overlapping user entries is provided. See `imsimta test -domain`.
3. If the base DN exists, substitute it into the URL in place of the $V.
4. Any applicable hosted domain for this entry is now determined. This is done by comparing either the canonical domain (if bit 2 (value = 4) of `DOMAIN_UPLEVEL` is clear) or the current domain (if bit 2 (value = 4) of `DOMAIN_UPLEVEL` is set) with the `service.defaultdomain configutil` parameter. If they do not match, the entry is a member of a hosted domain. The `service.defaultdomain configutil` parameter can be overridden by setting the `LDAP_DEFAULT_DOMAIN` MTA option in the `option.dat` file.
5. If the base DN determination fails, remove a component from the left hand side of the domain and go to Step 1. The substitution fails if no components remain.

$V also accepts an optional numeric argument. If it is set to 1 (for example, $1V), a failure to resolve the domain in the domain tree will be ignored and the base of the user tree specified by the `local.ugldapbasedn configutil` option will be returned.

If the attempt to retrieve the domain's base DN succeeds, the MTA also retrieves several useful domain attributes which will be needed later. The names of the attributes retrieved are set by the following MTA options in the `option.dat` file:

- `LDAP_DOMAIN_ATTR_UID_SEPARATOR` (default `domainUidSeparator`)
- `LDAP_DOMAIN_ATTR_SMARTHOST` (default `mailRoutingSmartHost`)
- `LDAP_DOMAIN_ATTR_CATCHALL_ADDRESS` (default `mailDomainCatchallAddress`)
- `LDAP_DOMAIN_ATTR_CATCHALL_MAPPING` (no default value)
- `LDAP_DOMAIN_ATTR_BLOCKLIMIT` (default `mailDomainMsgMaxBlocks`)
- `LDAP_DOMAIN_ATTR_REPORT_ADDRESS` (default `mailDomainReportAddress`)

- LDAP_DOMAIN_ATTR_STATUS (default inetDomainStatus)
- LDAP_DOMAIN_ATTR_MAIL_STATUS (default mailDomainStatus)
- LDAP_DOMAIN_ATTR_CONVERSION_TAG (default mailDomainConversionTag)
- LDAP_DOMAIN_ATTR_FILTER (default mailDomainSieveRuleSource)
- LDAP_DOMAIN_ATTR_DISK_QUOTA (no default)
- LDAP_DOMAIN_ATTR_MESSAGE_QUOTA (no default)
- LDAP_DOMAIN_ATTR_AUTOREPLY_TIMEOUT (no default)
- LDAP_DOMAIN_ATTR_NOSOLICIT (no default)
- LDAP_DOMAIN_ATTR_OPTIN (no default)
- LDAP_DOMAIN_ATTR_RECIPIENTLIMIT (no default)
- LDAP_DOMAIN_ATTR_RECIPIENTCUTOFF (no default)
- LDAP_DOMAIN_ATTR_SOURCEBLOCKLIMIT (no default)

Calling a Mapping From a URL

There might be unusual cases where the mapping from domain to base DN is done some other way. In order to accommodate such setups, the URL resolution process has the ability to call an MTA mapping. This is done with a metacharacter sequence of the general form:

$ | */mapping-name/mapping-argument* |

The double quotation (") initiates and terminates the callout. The character immediately following the $ is the separator between the mapping name and argument; a character should be chosen that does not collide with the expected character values used in either the mapping name or argument.

The $R Metacharacter

The $R metacharacter provides an appropriate filter for the URL. The intent is to produce a filter that searches all the attributes that might contain an email address for a particular user or group. The list of attributes to search comes from the configutil parameter local.imta.mailaliases. If this parameter is not set, the local.imta.schematag configutil parameter is examined, and depending on its value, an appropriate set of default attributes is chosen as follows:

sims401 mail, rfc822mailalias

nms41 mail,mailAlternateAddress

ims50 mail, mailAlternateAddress, mailEquivalentAddress

The value of local.imta.schematag can be a comma-separated list. If more than one schema is supported, the combined list of attributes with duplicates eliminated is used. The LDAP_SCHEMATAG MTA option can be used to override the setting of local.imta.schematag specifically for the MTA.

Additionally, the filter searches not only for the address that was originally supplied, but also for an address with the same local part but the domain that was actually found in the domain tree, which was saved in the second step in The $V Metacharacter. The iterative nature of the domain tree lookup means the two addresses may be different. This additional check is controlled by bit 1 (value = 2) of the DOMAIN_UPLEVEL MTA option in the option.dat file. Setting the bit enables the additional address check. The default value of DOMAIN_UPLEVEL is 0.

For example, suppose that the domain siroe.com appears in the domain tree. Assuming Oracle LDAP Schema 1 is in force, a lookup of the address:

u@host1.siroe.com

The filter that results from the expansion of $R and an ims50 schematag looks like:

```
(|(mail=u@siroe.com)
    (mail=u@host1.siroe.com)
    (mailAlternateAddress=u@siroe.com)
    (mailAlternateAddress=u@host1.siroe.com)
    (mailEquivalentAddress=u@siroe.com)
    (mailEquivalentAddress=u@host1.siroe.com))
```

If, on the other hand, DOMAIN_UPLEVEL was set to 1 rather than 3, the filter would be:

```
(|(mail=u@host1.siroe.com)
       (mailAlternateAddress=u@host1.siroe.com)
       (mailEquivalentAddress=u@host1.siroe.com))
```

Determining the Attributes to Fetch

If the URL specifies an asterisk (*) for the list of attributes to return, the asterisk is replaced with the list of attributes the MTA is able to use. This list is dynamically generated from the various MTA option settings that specify the options the MTA consumes.

Handling LDAP Errors

At this point the resulting URL is used to perform an LDAP search. If an LDAP error of some kind occurs, processing terminates with a temporary failure indication (4xx error in SMTP). If the LDAP operation succeeds but fails to produce a result, the catchall address attribute for the domain retrieved from the LDAP_DOMAIN_ATTR_CATCHALL_ADDRESS MTA option is checked. If it is set, its value replaces the current address.

If no catchall address attribute is set, the smarthost attribute for the domain retrieved from the LDAP_DOMAIN_ATTR_SMARTHOST MTA option is checked. If it is set, an address of the form

@smarthost:*user@domain*

is created and alias processing terminates successfully with this result. Additionally, the conversion tag for the domain obtained from the LDAP_DOMAIN_ATTR_CONVERSION_TAG MTA option will, if present, be attached to the address so that conversions can be done prior to forwarding to the smarthost. If no catchall address or smarthost exists for the domain, processing of this alias URL terminates unsuccessfully.

Sanity Checks on the LDAP Result

After the LDAP search has returned a result, it is checked to verify that there is only one entry in it. If there are more than one, each entry is checked to see if it has the right object class for a user or a group, a non-deleted status, and for users, a UID. Entries that do not pass this check are ignored. If the list of multiple entries is reduced to one by this check, processing proceeds. If not, a duplicate or ambiguous directory error is returned.

Support for Vanity Domains

The ALIAS_URL0 check is for a conventional user or a user in a hosted domain. If this fails a vanity domain check is also made. This is done with the following alias URL:

```
ALIAS_URL1=ldap:///$B?*?sub?(&(msgVanityDomain=$D)$R)
```

Support for Catchall Addresses

Finally, a check for a catchall address of the form *@host* needs to be made in the `mailAlternateAddress` attribute. This form of wildcarding is allowed in both hosted and vanity domains, so the proper alias URL for it is:

```
ALIAS_URL2=ldap:///$1V?*?sub?(mailAlternateAddress=@$D)
```

Note
The +* subaddress substitution mechanism has always worked with catch-all addresses in direct LDAP mode, but the string that was substituted was the subaddress only, not the entire local part. This has been changed so that the entire local part of the original address will be plugged into the catch-all address as a subaddress when this construct is used.

For example, given an address of the form `foo+bar@domain.com`, no local user `foo` in the `domain.com` domain, and a catch-all address for `domain.com` of `bletch+*@example.com`, the resulting address will now be `bletch+foo+bar@example.com`. It used to be `bletch+bar@example.com`.

Processing the LDAP Result

LDAP alias result processing is done in a number of order-dependent stages. These stages are described in the following sections:

- Object Class Check
- Entry Status Checks
- UID Check
- Message Capture
- Seeding the Reversal Cache
- Mail Host and Routing Address
- Miscellaneous Attribute Support
- Delivery Options Processing
- Additional Metacharacters for Use in Delivery Options
- Delivery Option Defaults
- Start and End Date Checks
- Optin and Presence Attributes
- Sieve Filter Handling
- Deferred Processing Control
- Group Expansion Attributes

Object Class Check

If the alias search succeeds, the object class of the entry is checked to make sure it contains an appropriate set of object classes for a user or a group. The possible sets of required object classes for users and groups is normally determined by what schemata are active. This is determined by the `local.imta.schematag` setting.

The following tables shows the user and group object classes that result from various `schematag`

values.

Object Classes Resulting from Various `schematag` Values

schematag	User Object Classes	Group Object Classes
sims40	`inetMailRouting+inetmailuser`	`inetMailRouting+inetmailgroup`
nms41	`mailRecipient + nsMessagingServerUser`	`mailGroup`
ims50	`inetLocalMailRecipient+inetmailuser`	`inetLocalMailRecipient + inetmailgroup`

The information in this table, like the rest of schema tag handling, is hard coded. However, there are also two MTA options in the `option.dat` file, `LDAP_USER_OBJECT_CLASSES` and `LDAP_GROUP_OBJECT_CLASSES`, which can be set to specify different sets of object classes for users and groups respectively.

For example, a schema tag setting of `ims50,nms41` would be equivalent to the following option settings:

```
LDAP_USER_OBJECT_CLASSES=inetLocalMailRecipient+inetmailuser,
mailRecipient+nsMessagingServerUser

LDAP_GROUP_OBJECT_CLASSES=inetLocalMailRecipient+inetmailgroup, mailGroup
```

The LDAP result is simply ignored if it does not have a correct set of object classes appropriate for a user or a group. The MTA also determines if it is dealing with a user or a group and saves this information. This saved information will be used repeatedly later.

Note that the object class settings described here are also used to construct an actual LDAP search filter that can be used to check to see that an entry has the right object classes for a user or a group. This filter is accessible through the `$K` metacharacter. It is also stored internally in the MTA's configuration for use by channel programs and is written to the MTA option file, `option.dat`, as the `LDAP_UG_FILTER` option when the command `imsimta cnbuild -option` is used. This option is only written to the file. The MTA never reads it from the option file.

Entry Status Checks

Next the entry's status is checked. There are two status attributes, one for the entry in general and another specifically for mail service.

The following table shows the general and mail-specific user or group attributes in the schema tag entry to check against depending on what schema tag are in effect

Attributes to Check Against

schematag	Type	General	Mail-specific
sims40	users	`inetsubscriberstatus`	`mailuserstatus`
sims40	groups	none	`inetmailgroupstatus`
nms41	users	none	`mailuserstatus`
nms41	groups	none	none
Messaging Server 5.0	users	`inetuserstatus`	`mailuserstatus`
Messaging Server 5.0	groups	none	`inetmailgroupstatus`

If necessary the LDAP_USER_STATUS and LDAP_GROUP_STATUS MTA options in the option.dat file can be used to select alternate general status attributes for users and groups respectively. The mail-specific user and group status attributes are controlled by the LDAP_USER_MAIL_STATUS and LDAP_GROUP_MAIL_STATUS MTA options.

Another factor that plays into this are the statuses for the domain itself (LDAP_DOMAIN_ATTR_STATUS and LDAP_DOMAIN_ATTR_MAIL_STATUS). All in all there are four status attributes. They are combined by considering them in the following order:

1. Domain status
2. Domain mail status
3. User or group status
4. Mail user or group status

The first of these that specifies something other than "active" takes precedence over all the others. The other permissible status values are "inactive", "deleted", "removed", "disabled", "hold", and "overquota". “Hold,” “disabled,” and “removed” statuses may only be specified for mail domains, mail users, or mail groups. “Overquota” status can only be specified as a mail domain or mail user status.

All statuses default to “active” if a particular status attribute is not present. Unknown status values are interpreted as “inactive.”

When the four statuses are combined, the following statuses for a user or group are possible: "active", "inactive", "deleted", "removed", "disabled", "hold", and "overquota." Active status causes alias processing to continue. Inactive or overquota status results in immediate rejection of the address with a 4xx (temporary) error. Deleted, removed, and disabled statuses results in immediate rejection of the address with a 5xx (permanent) error. Hold status is treated as active as far as status handling is concerned but it sets an internal flag so that when delivery options are considered later on, any options that are there are overridden with an option list containing a single "hold" entry.

UID Check

The next step is to consider the entry's UID. UIDs are used for a variety of purposes and must be part of all user entries and may be included in group entries. A user entry without a UID is ignored and processing of this alias URL terminates unsuccessfully. UIDs for entries in a hosted domain can consist of the real UID, a separator character, and then a domain. The MTA only wants the real UID so the rest is removed if present using the domain separator character obtained with the LDAP_DOMAIN_ATTR_UID_SEPARATOR MTA option in the option.dat file.

In the unlikely event that an attribute other than uid is used to store UIDs, the LDAP_UID MTA option can be used to force use of a different attribute.

Message Capture

Next the LDAP attribute used to specify one or more message capture addresses is checked. The attribute used for this purpose must be specified with the LDAP_CAPTURE MTA option. There is no default. Values of this attribute are treated as addresses and a special “capture” notification is generated and sent to these address containing the current message as an attachment. Additionally, the capture addresses are used to seed the address reversal cache in the likely event the address will subsequently appear as an envelope from: address.

Seeding the Reversal Cache

Next the primary address and any aliases attached to the user entry are considered. This information is used to seed the address reversal cache. It plays no part in the current address translation process. First, the primary address, personal name, recipient limit, recipient cutoff, and source block limit attributes are considered. The primary address is normally stored in the “mail” attribute; another attribute can be specified by setting the LDAP_PRIMARY_ADDRESS MTA option appropriately. (The primary address

reverses to itself, of course.) There is no default attribute for any of the other attributes. If you want to use them, you must specify them with the LDAP_PERSONAL_NAME, (see Vacation Autoreply Attributes) LDAP_RECIPIENTLIMIT, LDAP_RECIPIENTCUTOFF, and LDAP_SOURCEBLOCKLIMIT MTA options. The corresponding domain-level recipient limit, recipient cutoff, and source block limit attributes are also considered at this point. User-level settings completely override any domain-level setting.

Next, any secondary addresses are considered and a cache entry is made for each one. There are two sorts of secondary addresses: Those that undergo address reversal and those that do not. Both must be considered in order to properly seed the address reversal cache because of the need to check for message capture requests in all cases.

Secondary addresses that undergo reversal are normally stored in the mailAlternateAddress attribute. Another attribute can be specified by setting the LDAP_ALIAS_ADDRESSES MTA option. Secondary addresses that do not undergo reversal are normally stored in the mailEquivalentAddress attribute. Another attribute can be specified with the LDAP_EQUIVALENCE_ADDRESSES MTA option.

Mail Host and Routing Address

It is now time to consider the mailhost and mailRoutingAddress attributes. The actual attributes considered can be overridden with the LDAP_MAILHOST and LDAP_ROUTING_ADDRESS MTA options, respectively. These attributes work together to determine whether or not the address should be acted on at this time or forwarded to another system.

The first step is to decide whether or not mailhost is meaningful for this entry. A preliminary check of the delivery options active for the entry is done to see whether or not the entry is mailhost-specific. If it is not, mailhost checking is omitted. See the description of Delivery Options Processing, and the # flag in particular, to understand how this check is done.

In the case of a user entry, the mailhost attribute must identify the local system in order to be acted on. The mailhost attribute is compared with the value of the local.hostname configutil parameter and against the list of values specified by the local.imta.hostnamealiases configutil parameter. The mailhost attribute is deemed to identify the local host if any of these match.

A successful match means that the alias can be acted on locally and alias processing continues. An unsuccessful match means that the message needs to be forward to the mailhost to be acted on. A new address of the form:

@mailhost:*user@domain*

is constructed and becomes the result of the alias expansion operation.

The handling of a missing mailhost attribute is different depending on whether the entry is a user or a group. In the case of a user, a mailhost is essential, so if no mailhost attribute is present a new address of the form

@smarthost:*user@domain*

is constructed using the smart host for the domain determined by the LDAP_DOMAIN_ATTR_SMARTHOST MTA option. An error is reported if no smart host exists for the domain.

Groups, on the other hand, do not require a mailhost, so a missing mailhost is interpreted as meaning that the group can be expanded anywhere. So alias processing continues.

The mailRoutingAddress attribute adds one final wrinkle. Its presence causes processing to terminate with the mailRoutingAddress as the result. In Messaging Server 5.2, the mailHost check was done first and must pass before the routing address can take effect. To get the same behavior in the current version of Messaging Server, the format of the mailRoutingAddress attribute could be as follows: mailRoutingAddess: @mailhost:user@domain

Miscellaneous Attribute Support

Next the `mailMsgMaxBlocks` attribute is considered. First it is minimized with the domain block limit returned from the `LDAP_DOMAIN_ATTR_BLOCKLIMIT` MTA option. If the size of the current message is known to exceed the limit, alias processing terminates with a size-exceeded error. If the size is not known or does not exceed the limit, the limit is nevertheless stored and will be rechecked when the message itself is checked later. The use of `mailMsgMaxBlocks` can be overridden with the `LDAP_BLOCKLIMIT` MTA option.

Next a number of attributes are accessed and saved. Eventually these will be written into the queue file entry for use by the `ims_master` channel program, which will then use them to update the store's user information cache. If the attributes are not found for individual users, domain-level attributes can be used to set defaults.

This step is skipped if the LDAP entry is for a group rather than a user or if the LDAP entry came from the alias cache and not from the LDAP directory. The logic behind the latter criteria is that frequent updates of this information are unnecessary and using the alias cache offers a reasonable criteria for when updates should be done. The names of the attributes retrieved are set by various MTA options.

The following table shows the MTA options which set the retrieved disk quota and message quota attributes.

MTA Options Which Set the Retrieved Disk Quota and Message Quota Attributes

MTA option	Attribute
`LDAP_DISK_QUOTA`	`mailQuota`
`LDAP_MESSAGE_QUOTA`	`mailMsgQuota`

Next a number of attributes are stored for possible use in conjunction with metacharacter substitutions later.

The following table shows the MTA options, the default attribute, and metacharacters.

MTA Options, Default Attributes, and Metacharacters

MTA Option	Default Attribute	Metacharacters
`LDAP_PROGRAM_INFO`	`mailProgramDeliveryInfo`	`$P`
`LDAP_DELIVERY_FILE`	`mailDeliveryFileURL`	`$F`
`LDAP_SPARE_1`	no default	`$1E $1G $E`
`LDAP_SPARE_2`	no default	`$2E $2G $G`
`LDAP_SPARE_3`	no default	`$3E $3G`
`LDAP_SPARE_4`	no default	`$4E $4G`
`LDAP_SPARE_5`	no default	`$5E $5G`

Spare slots for additional attributes are included so that you can use them to build customized address expansion facilities.

Next any values associated with the `mailconversiontag` attribute are added to the current set of conversion tags. The name of this attribute can be changed with the `LDAP_CONVERSION_TAG` MTA option. If any values were associated with the domain's `mailDomainConversionTag` attribute, they are attached as well.

Delivery Options Processing

Next the `mailDeliveryOption` attribute is checked. The name of this attribute can be changed with the `LDAP_DELIVERY_OPTION` MTA option. This is a multi valued option and its values determine the addresses produced by the alias translation process. Additionally, the permissible values are different for users and groups. Common permissible values are `program`, `forward`, and `hold`." User-only values are `mailbox`, `native`, `unix`, and `autoreply`. The group-only values are `members`, `members_offline`, and `file`.

The conversion of the mailDeliveryOption attribute into appropriate addresses is controlled by the `DELIVERY_OPTIONS` MTA option. This option not only specifies what addresses are produced by each permissible `mailDeliveryOption` value, but also what the permissible `mailDeliveryOption` values are and whether or not each one is applicable to users, groups, or both.

The value of this option consists of a comma-separated list of `deliveryoption=template` pairs, each pair with one or more optional single character prefixes.

The default value of the `DELIVERY_OPTIONS` option is:

```
DELIVERY_OPTIONS=*mailbox=$M%$\$2I$_+$2S@ims-ms-daemon, \
     &members=*,                                        \
     *native=$M@native-daemon,                          \
     /hold=@hold-daemon:$A,                             \
     *unix=$M@native-daemon,                            \
     &file=+$F@native-daemon,                           \
     &@members_offline=*,                               \
     program=$M%$P@pipe-daemon,                         \
     #forward=**,                                       \
     *^!autoreply=$M+$D@bitbucket
```

Each delivery option corresponds to a possible `mailDeliveryOption` attribute value and the corresponding template specifies the resulting address using the same metacharacter substitution scheme used by URL processing.

The following table shows the single character prefixes available for the `DELIVERY_OPTIONS` options.

Single-Character Prefixes for Options in the DELIVERY_OPTIONS MTA Option

Character Prefix	Description
@	Sets a flag saying that the message needs to be redirected to the reprocess channel. Processing of the current user/group is abandoned. Flag ignored for messages originating from the reprocess channel.
*	Delivery option applies to users.
&	Delivery option applies to groups.
$	Sets a flag saying expansion of this user or group is to be deferred.
^	Sets a flag saying that the vacation start and end times should be checked to see if this delivery option really is in effect.
#	Sets a flag saying expansion of this delivery option does not need to take place on the entry's designated mailhost. That is, the following entry is mailhost-independent. This lets the MTA check to see if all of a given user or group's delivery options are independent of the mailhost. If this condition is satisfied the MTA can act on the entry immediately rather than having to forward the message to the mailhost.
/	Sets a flag that causes all addresses produced by this delivery option to be held. Message files containing these recipient addresses will have a `.HELD` extension.
!	Sets a flag that says that autoreply operations should be handled internally by the MTA. It only makes sense to use this prefix on an autoreply delivery option. The option's value should direct the message to the bitbucket channel.

If neither * nor & are present, the delivery option is taken to apply to both users and groups.

Additional Metacharacters for Use in Delivery Options

Several additional metacharacters have been added to support this new use of the MTA's URL template facility. These include:

The following table shows additional metacharacters and their descriptions for use in delivery options.

Additional Metacharacters for Use in Delivery Options

Metacharacter	Description
$	Force subsequent text to lower case.
$^	Force subsequent text to upper case.
$_	Perform no case conversion on subsequent text.
$*n*A	Insert the _n_th character of the address. The first character is character 0. The entire address is substituted if *n* is omitted. This is intended to be used to construct autoreply directory paths.
$D	Insert the domain part of the address.
$*n*E	Insert the value of the _n_th spare attribute. If *n* is omitted the first attribute is used.
$F	Insert the name of the delivery file (`mailDeliveryFileURL` attribute).
$*n*G	Insert the value of the _n_th spare attribute. If *n* is omitted the second attribute is used.
$*n*H	Insert the _n_th component of the domain from the original address counting from 0. The default is 0 if *n* is omitted.
$*n*I	Insert hosted domain associated with alias. This metacharacter accepts an integer parameter *n* whose semantics are described in Integers Controlling Behavior Modification of the $nI and $nS Metacharacters.
$*n*J	Insert the _n_th part of the host domain counting from 0. The default for *n* is 0.
$_n_O	Insert source route associated with the current address. This metacharacter accepts an integer parameter n whose semantics are described in Integers Controlling Behavior Modification of the $nI and $nS Metacharacters.
$K	Insert an LDAP filter that matches the object classes for a user or group. See the description of the `LDAP_UG_FILTER` output-only MTA option.
$L	Insert the local part of the address.
$*n*M	Insert the _n_th character of the UID. The first character is character 0. The entire UID is substituted if n is omitted.
$P	Insert the program name (`mailProgramDeliveryInfo` attribute).
$*n*S	Insert subaddress associated with the current address. This metacharacter accepts an integer parameter n whose semantics are described in Integers Controlling Behavior Modification of the $nI and $nS Metacharacters.
$*n*U	Insert the nth character of the dequoted form of the mailbox part of the current address. The first character is character 0. The entire dequoted mailbox is substituted if n is omitted.
$*n*X	Insert the _n_th component of the mailhost. The entire mailhost is inserted if *n* is omitted.

The following table shows how the integer parameter modifies the behavior of the $nI and $nS metacharacters.

Integers Controlling Behavior Modification of the $nI and $nS Metacharacters

Integer	Description of Behavior
0	Fail if no value is available (default).
1	Insert value if one is available. Insert nothing if not.
2	Insert value if one is available. Insert nothing and delete preceding character is one is not (this particular behavior is needed by the ims-ms channel).
3	Insert value if one is available. Insert nothing and ignore following character if one is not.

In addition to the metacharacters, the following table shows two special template strings.

Special Template Strings

Special Template String	Description
*	Perform group expansion. This value is not valid for user entries.
**	Expand the attribute named by the LDAP_FORWARDING_ADDRESS MTA option. This defaults to mailForwardingAddress.

With group expansion, for example, if a user's mailDeliveryOption value is set to mailbox, we form a new address consisting of the stripped UID, a percent sign followed by the hosted domain if one is applicable, a plus sign followed by the subaddress if one was specified, and finally @ims-ms-daemon.

Delivery Option Defaults

If the list of active delivery options is empty at this point, the first option on the list (usually mailbox) is activated for users and the second option on the list (usually members) is activated for groups.

Start and End Date Checks

Start and end dates are checked after the delivery option list has been read. There are two attributes whose names are controlled by the LDAP_START_DATE (default vacationStartDate) and LDAP_END_DATE (default vacationEndDate) MTA options, respectively. If one or more of the active delivery options specified the ^ prefix character, the values of these options are checked against the current date. If the current date is outside the range specified by these options, the delivery options with the ^ prefix are removed from the active set. For more information see Vacation Autoreply Attributes.

Optin and Presence Attributes

The LDAP_OPTIN1 through LDAP_OPTIN8 MTA options specify LDAP attributes for per-user spam filter opt-in values based on destination addresses. If an option is specified and the attribute is present, it is appended to the current spam filter opt-in list. Any values set by the domain level attribute set by the LDAP_DOMAIN_ATTR_OPTIN MTA option will also be appended to the list. LDAP_SOURCE_OPTIN1 through LDAP_SOURCE_OPTIN8 provide comparable originator-address-based per-user spam filter optins.

The LDAP_PRESENCE MTA option can be used to specify a URL that can be resolved to return presence information about the user. If the option is specified and the attribute is present, its value is saved for possible use in conjunction with Sieve presence tests. The domain level attribute set by the LDAP_DOMAIN_ATTR_PRESENCE MTA option is used as source for this URL if no value exists for the user entry.

Sieve Filter Handling

Next the mailSieveRuleSource attribute is checked for a Sieve filter that applies to this entry. If this attribute exists, it is parsed and stored at this point. The two possible forms for the value of this attribute are a single value that contains a complete Sieve script or multiple values where each value contains a piece of a Sieve script. The latter form is produced by the web filter construction interface. Special code is used to order the values and glue them together properly.

The use of the mailSieveRuleSource attribute specifically can be overridden by using the LDAP_FILTER MTA option.

Deferred Processing Control

Next the mailDeferProcessing attribute is checked. This attribute can be changed by using the LDAP_REPROCESS MTA option. Processing continues normally if this attribute exists and is set to no. But if this attribute is set to yes and the current source channel is not the reprocess channel, expansion of this entry is aborted and the original *user@domain* address is simply queued to the reprocess channel. If this attribute does not exist, the setting of the deferred processing character prefix associated with delivery options processing is checked. (See the section Delivery Options Processing the default for users is no. The default for groups is controlled by the MTA option DEFER_GROUP_PROCESSING, which defaults to 1 (yes). Alias processing concludes at this point for user entries.

Group Expansion Attributes

A number of additional attributes are associated with group expansion and must be dealt with at this point. The names of these attributes are all configurable via various MTA options.

The following table lists the default attribute names, the MTA option to set the attribute name, and the way the attribute is processed by the MTA. The ordering of the elements in the table shows the order in which the various group attributes are processed. This ordering is essential for correct operation.

Group Expansion Default Attributes and MTA Option to Set

Default Attribute	(MTA option to Set Attribute Name) How the Attribute is Processed
mgrpMsgRejectAction	(LDAP_REJECT_ACTION) Single valued attribute that controls what happens if any of the subsequent access checks fail. Only one value is defined: TOMODERATOR, which if set instructs the MTA to redirect any access failures to the moderator specified by the mgrpModerator attribute. The default (and any other value of this attribute) causes an error to be reported and the message rejected.
mailRejectText	(LDAP_REJECT_TEXT) The first line of text stored in the first value of this attribute is saved. This text will be returned if any of the following authentication attributes cause the message to be rejected. This means the text can appear in SMTP responses so value has to be limited to US-ASCII to comply with current messaging standards.

mgrpBroadcasterPolicy	(LDAP_AUTH_POLICY) Specifies level of authentication needed to send to the group. Possible tokens are SMTP_AUTH_REQUIRED or AUTH_REQ, both of which mean that the SMTP AUTH command must be used to identify the sender in order to send to the group; SMTP_AUTH_USED and AUTH_USED which are similar in effect to SMTP_AUTH_REQUIRED and AUTH_REQ, but do not require posters to authenticate; PASSWORD_REQUIRED, PASSWD_REQUIRED, or PASSWD_REQ, all of which mean the password to the list specified by the mgrpAuthPassword attribute must appear in an Approved: header field in the message; OR, which changes the OR_CLAUSES MTA option setting to 1 for this list; AND, which changes the OR_CLAUSES MTA option setting to 0 for this list; and NO_REQUIREMENTS, which is non-operational. Multiple values are allowed and each value can consist of a comma-separated list of tokens.If SMTP AUTH is called for it also implies that any subsequent authorization checks will be done against the email address provided by the SASL layer rather than the MAIL FROM address.
mgrpAllowedDomain	(LDAP_AUTH_DOMAIN) Domains allowed to submit messages to this group. A match failure with the OR_CLAUSES MTA option set to 0 (the default) means access checking has failed and all subsequent tests are bypassed. A match failure with the OR_CLAUSES MTA option set to 1 sets a “failure pending” flag; some other access check must succeed in order for access checking to succeed. This check is bypassed if the submitter has already matched an LDAP_AUTH_URL. Can be multivalued and glob-style wildcards are allowed.
mgrpDisallowedDomain	(LDAP_CANT_DOMAIN) Domains not allowed to submit messages to this group. A match means access checking has failed and all subsequent checks are bypassed. This check is bypassed if the submitter has already matched an LDAP_AUTH_URL. Can be multivalued and glob-style wildcards are allowed.
mgrpAllowedBroadcaster	(LDAP_AUTH_URL) URL identifying mail addresses allowed to send mail to this group. Can be multivalued. Each URL is expanded into a list of addresses and each address is checked against the current envelope from: address. A match failure with the OR_CLAUSES MTA option set to 0 (the default) means access checking has failed and all subsequent tests are bypassed. A match failure with the OR_CLAUSES MTA option set to 1 sets a "failure pending" flag. Some other allowed access check must succeed in order for access checking to succeed. A match also disables subsequent domain access checks. The expansion that is performed is similar to an SMTP EXPN with all access control checks disabled.List expansion in the context of the mgrpallowedbroadcaster LDAP attribute now includes all the attributes used to store email addresses (normally mail, mailAlternateAddress, and mailEquivalentAddress). Previously only mail attributes were returned, making it impossible to send to lists restricted to their own members using alternate addresses.

mgrpDisallowedBroadcaster	(LDAP_CANT_URL) URL identifying mail addresses not allowed to send mail to this group. Can be multivalued. Each URL is expanded into a list of addresses and each address is checked against the current envelope from: address. A match means access checking has failed and all subsequent checks are bypassed. The expansion that is performed is similar to an SMTP EXPN with all access control checks disabled.
mgrpMsgMaxSize	(LDAP_ATTR_MAXIMUM_MESSAGE_SIZE) Maximum message size in bytes that can be sent to the group. This attribute is obsolete but still supported for backwards compatibility; the new mailMsgMaxBlocks attribute should be used instead.
mgrpAuthPassword	(LDAP_AUTH_PASSWORD) Specifies a password needed to post to the list. The presence of a mgrpAuthPassword attribute forces a reprocessing pass. As the message is enqueued to the reprocessing channel, the password is taken from the header and placed in the envelope. Then, while reprocessing, the password is taken from the envelope and checked against this attribute. Additionally, only passwords that actually are used are removed from the header field.The OR_CLAUSES MTA option acts on this attribute in the same way it acts on the other access check attributes.
mgrpModerator	(LDAP_MODERATOR_URL) The list of URLs given by this attribute to be expanded into a series of addresses. The interpretation of this address list depends on the setting of the LDAP_REJECT_ACTION MTA option. If LDAP_REJECT_ACTION is set to TOMODERATOR, this attribute specifies the moderator address(es) the message is to be sent to should any of the access checks fail. If LDAP_REJECT_ACTION is missing or has any other value, the address list is compared with the envelope from address. Processing continues if there is a match. If there is no match, the message is again sent to all of the addresses specified by this attribute. Expansion of this attribute is implemented by making the value of this attribute the list of URLs for the group. Any list of RFC822 addresses or DNs associated with the group is cleared, and the delivery options for the group are set to members. Finally, subsequent group attributes listed in this table are ignored.
mgrpDeliverTo	(LDAP_GROUP_URL1) List of URLs which, when expanded, provides a list of mailing list member addresses.
memberURL	(LDAP_GROUP_URL2) Another list of URLs which, when expanded, provides another list of mailing list member addresses.
uniqueMember	(LDAP_GROUP_DN) List of DNs of group members. DNs may specify an entire subtree. Unique member DNs are expanded by embedding them in an LDAP URL. The exact URL to use is specified by the GROUP_DN_TEMPLATE MTA option. The default value for this option is: ldap:///$A??sub?mail=*$A specifies the point where the uniqueMember DN is inserted.
mgrpRFC822MailMember	(LDAP_GROUP_RFC822) Mail addresses of members of this list.
rfc822MailMember	(LDAP_GROUP_RFC822) rfc822MailMember is supported for backwards compatibility. Either rfc822MailMember or mgrpRFC822MailMember, but not both, can be used in any given group.

mgrpErrorsTo	(LDAP_ERRORS_TO) Sets the envelope originator (MAIL FROM) address to whatever the attribute specifies.
mgrpAddHeader	(LDAP_ADD_HEADER) Turns the headers specified in the attribute into header trimming ADD options.
mgrpRemoveHeader	(LDAP_REMOVE_HEADER) Turns the headers specified into header trimming MAXLINES=-1 options.
mgrpMsgPrefixText	(LDAP_PREFIX_TEXT) Adds the specified text to the beginning of the message text, if any.
mgrpMsgSuffixText	(LDAP_SUFFIX_TEXT) Adds the specified text to the ending of the message text, if any.
No Default	(LDAP_ADD_TAG) Checks the subject for the specified text; if it isn't present the text is added at the beginning of the subject field.

One final attribute is checked in the special case of group expansion as part of an SMTP EXPN command: mgmanMemberVisibility or expandable. The LDAP_EXPANDABLE MTA option can be used to select different attributes to check. Possible values are: anyone, which means that anyone can expand the group, all or true, which mean that the user has to successfully authenticate with SASL before expansion will be allowed, and none, which means that expansion is not allowed. Unrecognized values are interpreted as none. If the attribute is not present, the EXPANDABLE_DEFAULT MTA option controls whether the expansion is allowed.

Alias entries are cached in a fashion similar to domain entries. The MTA options controlling the alias cache are ALIAS_ENTRY_CACHE_SIZE (default 1000 entries) and ALIAS_ENTRY_CACHE_TIMEOUT (default 600 seconds). The entire LDAP return value for a given alias is retained in the cache.

Negative caching of alias entries is controlled by the ALIAS_ENTRY_CACHE_NEGATIVE MTA option. A nonzero value enables caching of alias match failures. A zero value disables it. Negative caching of alias entries is disabled by default. The theory is that repeated specification of an invalid address is unlikely to occur very often in practice. In addition, negative caching may interfere with timely recognition of new users added to the directory. However, sites should consider re enabling negative caching of aliases in situations where vanity domains are heavily used. The search performed by the URL specified in ALIAS_URL0 is less likely to be successful.

To Modify Group Membership Attribute Syntax

Support has been added for postprocessing LDAP expansion results with a mapping. The new LDAP_URL_RESULT_MAPPING MTA option can be used to specify the name of a group attribute which in turn specifies the name of a mapping. This mapping will be applied to any results returned by expanding either a mgrpDeliverTo or memberURL attribute. The mapping probe will be of the form:

LDAP-URL | *LDAP-result*

If the mapping returns with $Y set the mapping result string will replace the LDAP result for alias processing purposes. If the mapping returns with $N set the result will be skipped.

This mechanism can be used to define groups based on attributes that don't contain proper email address. For example, suppose a company has placed pager numbers in all their user entries. Messages can be sent to these numbers via email by suffixing them with a particular domain. A group could then be defined as follows:

1. Define a new mgrpURLResultMapping attribute in the directory and set the LDAP_URL_RESULT_MAPPING MTA option to this attribute's name.
2. Define a page-all group with the following attributes:

```
mgrpDeliverto: ldap:///o=usergroup?pagerTelephoneNumber?sub
mgrpURLResultMapping: PAGER-NUMBER-TO-ADDRESS
```

3. Define the mapping:

```
PAGER-NUMBER-TO-ADDRESS
  *|*    "$1"@pagerdomain.com$Y
```

Even more interesting effects can be achieved by combining this mechanism with the PROCESS_SUBSTITUTION mechanism described in Optimizing Authorization Checks to the LDAP Directory for Messages Addressed to Mailing Lists. For example, it would be easy to create a metagroup where sending to an address of the form:

pager+user@*domain.com*

sends a page to the user named user.

Address Reversal

Address reversal with direct LDAP starts with a USE_REVERSE_DATABASE value of 4, which disables the use of any reverse database. You should also set USE_TEXT_DATABASES to read the IMTA_TABLE:reverse.txt file, as the sleepycat databases are being deprecated. It then builds on the routing facilities previously discussed. In particular, in the previous version, it began with a reverse URL specification of the form:

REVERSE_URL=ldap:///$V?mail?sub?$Q

The $V metacharacter has already been described in the context of alias URLs However, the $Q metacharacter, while quite similar in function to the $R metacharacter used in alias URLs, is specifically intended for use in address reversal. Unlike $R, it produces a filter that searches attributes containing addresses that are candidates for address reversal. The list of attributes to search comes from the MTA option LDAP_MAIL_REVERSES. If this option is not set, the local.imta.schemetag configutil parameter is examined, and depending on its value, an appropriate set of default attributes is chosen.

Note
Changing REVERSE_URL for any reason is discouraged.

The following table shows the local.imta.schemetag values and the default attributes chosen.

local.imta.schemetag Values and Attributes

Schema Tag Value	Attributes
sims40	mail,rfc822mailalias
nms41	mail,mailAlternateAddress
ims50	mail,mailAlternateAddress

The use of $Q is no longer appropriate, however. In order for message capture and other facilities to work correctly, address reversal has been enhanced to pay attention to the attribute that matched in

addition to the fact that a match occurred. This means that $R should be used to specify the filter instead of $Q. Additionally, the $N metacharacter has been added, which returns a list of the attributes of interest to address reversal.

The value of $N cannot exactly be controlled: the MTA constructs it from its own, hard-coded (and subject to change) list of the relevant attributes for address reversal purposes. If you use the various LDAP_* global MTA options to change what the MTA thinks are the names of attributes of interest, you will, in fact, fetch different attributes from LDAP. But it is always whatever attributes that correspond semantically to the MTA's idea of relevant attributes. These are: LDAP_CAPTURE (no default), LDAP_RECIPIENTLIMIT (no default), LDAP_RECIPIENTCUTOFF (no default), LDAP_SOURCEBLOCKLIMIT (no default), LDAP_SOURCE_CHANNEL (no default), LDAP_PERSONAL_NAME (no default), LDAP_SOURCE_CONVERSION_TAG (no default), LDAP_PRIMARY_ADDRESS (mail), LDAP_ALIAS_ADDRESSES (mailAlternateAddress), LDAP_EQUIVALENCE_ADDRESSES (mailEquivalentAddress), plus the LDAP_SPARE_* attributes.

The resulting option value is:

```
REVERSE_URL=ldap:///$V?$N?sub?$R
```

As always, local.imta.schematag can be a comma-separated list. If more than one schema is supported, the combined list of attributes, with duplicates eliminated, is used.

Additionally, the filter searches not only for the address that was originally supplied, but also for an address with the same local part but the domain that was actually found in the domain tree (which was saved in step 2 of Rewrite Rule Machinery). The iterative nature of the domain tree lookup means the two addresses may be different.

For example, suppose that the domain siroe.com appears in the domain tree and the MTA looks the address:

```
u@host1.siroe.com
```

The filter that results from the expansion of $R and an ims50 schema tag will be something like:

```
(|(mail=u@siroe.com)
(mail=u@host1.siroe.com)
(mailAlternateAddress=u@siroe.com)
(mailAlternateAddress=u@host1.siroe.com)
(mailEquivalentAddress=u@siroe.com)
(mailEquivalentAddress=u@host1.siroe.com))
```

Reverse lookup returns several attributes, and the MTA knows to use the mail attribute (more precisely, the attribute named by LDAP_PRIMARY_ADDRESS) as the one for address reversal. Note that the mailEquivalentAddress (more precisely, the attribute named by LDAP_EQUIVALENCE_ADDRESSES) is also permitted.

After the URL is constructed an LDAP search is performed. If the search is successful, LDAP returns multiple attributes in essentially arbitrary order. Unsuccessful searches or errors leave the original address unchanged.

Due to the frequency with which address reversal operations are performed, especially given the number of addresses that can appear in a message header, and the expense of the directory queries involved, both negative and positive results need to be cached. This is implemented with an in-memory, open-chained, dynamically-expanded hash table. The maximum size of the cache is set by the REVERSE_ADDRESS_CACHE_SIZE MTA option (default 100000) and the timeout for entries in the cache is set by the REVERSE_ADDRESS_CACHE_TIMEOUT MTA option (default 600 seconds). The cache actually stores addresses themselves, not the LDAP URLs and LDAP results.

Asynchronous LDAP Operations

Asynchronous lookups avoid the need to store an entire large LDAP result in memory, which can cause performance problems in some cases. The MTA provides the ability to perform various types of lookups done by the MTA asynchronously.

Use of asynchronous LDAP lookups is controlled by the MTA option `LDAP_USE_ASYNC`. This option is a bit-encoded value. Each bit, if set, enables the use of asynchronous LDAP lookups in conjunction with a specific use of LDAP within the MTA.

The following table shows the bit and value settings for the `LDAP_USE_ASYNC` MTA option in the `option.dat` file.

Settings for the LDAP_USE_ASYNC MTA Option

Bit	Value	Specific Use of LDAP
0	1	`LDAP_GROUP_URL1` (`mgrpDeliverTo`) URLs
1	2	`LDAP_GROUP_URL2` (`memberURL`) URLs
2	4	`LDAP_GROUP_DN` (`UniqueMember`) DNs
3	8	`auth_list`, `moderator_list`, `sasl_auth_list`, and `sasl_moderator_list` nonpositional list parameter URLs
4	16	`cant_list`, `sasl_cant_list` nonpositional list parameter URLs
5	32	`originator_reply` nonpositional list parameter URLs
6	64	`deferred_list`, `direct_list`, `hold_list`, `nohold_list` nonpositional list parameter URLs
7	128	`username_auth_list`, `username_moderator_list`, `username_cant_list` nonpositional list parameter URLs
8	256	alias file list URLs
9	512	alias database list URLs
10	1024	`LDAP_CANT_URL` (`mgrpDisallowedBroadcaster`) outer level URLs
11	2048	`LDAP_CANT_URL` inner level URLs
12	4096	`LDAP_AUTH_URL` (`mgrpAllowedBroadcaster`) outer level URLs
13	8192	`LDAP_AUTH_URL` inner level URLs
14	16384	`LDAP_MODERATOR_URL` (`mgrpModerator`) URLs
15	32768	`LDAP_JETTTISON_URL` (`mgrpJettisonBroadcasters`) URLs (Introduced in Messaging Server 7 Update 3)

The default value of the `LDAP_USE_ASYNC` MTA option is 0, which means that asynchronous LDAP lookups are disabled by default.

Settings Summary

In order to enable direct LDAP, the following MTA options need to be set:

```
ALIAS_MAGIC=8764
ALIAS_URL0=ldap:///$V?*?sub?$R
USE_REVERSE_DATABASE=4
USE_DOMAIN_DATABASE=0
REVERSE_URL=ldap:///$V?$N?mail?sub?$R
```

If vanity domains are to be supported, the following additional options must be set:

```
DOMAIN_MATCH_URL=ldap:///$B?msgVanityDomain?sub? \
(msgVanityDomain=$D)
ALIAS_URL1=ldap:///$B?*?sub? (&(msgVanityDomain=$D)$R)
ALIAS_URL2=ldap:///$1V?*?sub?(mailAlternateAddress=@$D)
```

Note that the last of these options also handle the case of wild carded local parts in hosted as well as vanity domains. If wild carded local part support is desired but vanity domain support is not, the following option should be used instead:

```
ALIAS_URL1=ldap:///$V?*?sub?&(mailAlternateAddress=@$D)
```

The `filter ssrd:$A` clause needs to be removed from the ims-ms channel definition in the MTA configuration file (`imta.cnf`).

Processing Multiple Different LDAP Attributes with the Same Semantics

The MTA now has the ability to process multiple different LDAP attributes with the same semantics. Note that this is not the same as processing of multiple values for the same attribute, which has always been supported. The handling attributes receive depends on the semantics of the attribute. The possible options are:

1. Multiple different attributes don't make sense and render the user entry invalid. In Messaging Server 6.2 and later this handling is the default for all attributes unless otherwise specified.
2. If multiple different attribute are specified, one is chosen at random and used. `LDAP_AUTOREPLY_SUBJECT`, `LDAP_AUTOREPLY_TEXT`, and `LDAP_AUTOREPLY_TEXT_INT` all receive this handling in Messaging Server 6.2 only. In Messaging Server 6.3 and later they receive the handling described in Vacation Autoreply Attributes. Messaging Server 6.3 adds the `LDAP_SPARE_3` and `LDAP_PERSONAL_NAME` attribute to this category. Note that this was how all attributes were handled prior to Messaging Server 6.2.
3. Multiple different attributes do make sense and should all be acted on. This handling is currently in effect for `LDAP_CAPTURE`, `LDAP_ALIAS_ADDRESSES`, `LDAP_EQUIVALENCE_ADDRESSES` and `LDAP_DETOURHOST_OPTIN`. Note that `LDAP_DETOURHOST_OPTIN` attribute was first added to Messaging Server 6.3.

Chapter 8. About MTA Services and Configuration

About MTA Services and Configuration

This information describes general MTA services and configuration.

Topics:

Compiling the MTA Configuration

Whenever an MTA configuration file such as `imta.cnf`, `mappings`, `aliases`, or `option.dat` is modified, you must recompile the configuration. This compiles the configuration files into a single image in shared memory (on UNIX) or a dynamic link library (NT).

The compiled configuration has a static and dynamic reloadable part. If the dynamic part is changed, and you run an `imsimta reload`, a running program reloads the dynamic data. The dynamic parts are mapping tables, aliases, and lookup tables.

The main reason for compiling configuration information is performance. Another feature of using a compiled configuration is that configuration changes can be tested more conveniently since the configuration files themselves are not "live" when a compiled configuration is in use.

Whenever a component of the MTA (such as a channel program) must read the configuration file, it first checks to see if a compiled configuration exists. If it does, the image is attached to the running program. If the image attach operation fails, the MTA falls back on the old method of reading the text files instead.

If you make changes to the `reverse`, `forward`, or general databases, then issue the command `imsimta reload` to get the changes to take effect. If you make changes to the `imta.cnf`, `mappings` file, `aliases`, `conversions`, or `option.dat` files that do not affect the job controller, then you should issue an `imsimta cnbuild` followed by an `imsimta restart smtp`. If you make changes to `dispatcher.cnf` you need to do an `imsimta restart dispatcher`. If you make changes to the configuration files that are included in the compiled configuration that affect the job controller, but not the SMTP server, in many cases you should issue the following commands: `imsimta cnbuild` and `imsimta restart job_controller`.

If you make changes to the configuration files that are included in the compiled configuration that affect both the SMTP server and the job controller, you should issue the following commands:

```
imsimta cnbuild
imsimta restart smtp
imsimta restart job_controller
```

(See *Messaging Server Administration Reference* for details on these commands.)

Other instances where you must restart the job controller:

- Changing the controller configuration files, `job_controller.cnf` or `job_controller.site`, or any files included into job_controller.cnf.
- Adding or changing use of the channel keywords `pool`, `maxjobs`, `master`, `slave`, `single`, `single_sys`, or `multiple` in the `imta.cnf` file. Adding or changing a `threaddepth` channel keyword in the `imta.cnf` can be dealt with instead by running `imsimta cache -change -thread_depth=...`.
- If you want changes to master channel jobs to take effect immediately (rather than waiting for the controller to time-out existing channel jobs), then any relevant (which means almost all) changes to the MTA configuration or channel option files. (Changes to the `mappings` file or to MTA databases: (1) aren't usually relevant for outbound channel jobs, though they can matter to "intermediate" channels such as `conversion`, `process`, `reprocess`, and (2) if those intermediate channels are a concern, changes to the `mappings` file or to databases can often be handled by running `imsimta reload`, avoiding a restart of the Job Controller.) The desire to have changes take effect immediately needs to be balanced against the damage that restarting the Job Controller does. Also consider just how much longer a particular sort of job is going to run anyhow.

The MTA configuration includes `imta.cnf` and all files it includes (such as `internet.rules`), the `alias` file, the `mappings` file, the `conversions` file, the `option.dat` file (and any files any of the preceding include), and `imta.filter`, and the `reverse`, `forward`, and general data files, and potentially some `configutil` parameters.

Any changes (such as additions or changes to keywords on channel definitions) to `imta.cnf` also require running `imsimta cnbuild`. That's a basic, regardless of whether a Job Controller restart is needed.

Try to avoid restarting the Job Controller, especially at times of large numbers of messages in the queues, unless one of the preceding conditions necessitates a restart.

Do not use the `imsimta refresh` command on production systems because it is often not necessary to restart the job controller, and restarting the job controller resets message retries, delayed notification messages, bounced messages, and so on.

The MTA Configuration File

The primary MTA configuration file is `imta.cnf`. By default, this file is found at *msg-svr-base* `/config/imta.cnf`. This file contains MTA channel definitions as well as the channel rewrite rules. The channel associated with a rewritten destination address becomes the destination channel. The system will typically work well using the default `imta.cnf`.

This section provides a brief introduction to the MTA configuration file. For details about configuring the rewrite rules and channel definitions that make up the MTA configuration file, see Configuring Rewrite Rules and Configuring Channel Definitions.

By modifying the MTA configuration file, you establish the channels in use at a site and establish which channels are responsible for which sorts of addresses by using rewrite rules. The configuration file establishes the layout of the email system by specifying the transport methods available (channels) and

the transport routes (rewrite rules) associating types of addresses with appropriate channels.

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.

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. The rewrite rules appear in the upper half of the configuration file followed by the channel definitions in the lower half of the configuration file.

```
! 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.
!
! Part I: Rewrite rules
a     $U@a-daemon           (2)
b     $U@b-daemon
c     $U%c@b-daemon
d     $U%d@a-daemon
      (3)
! Part II: Channel definitions
l      (4)
local-host

a_channel defragment charset7 usascii      (5)
a-daemon

b_channel noreverse notices 1 2 3
b-daemon

</opt/SUNWmsgsr/msg-tango/table/internet.rules    (6)
```

The key items (labeled with 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. These definitions are collectively referred to as the_channel host table_, which defines the channels that the MTA can use and the names associated with each channel.
4. The first channel block to appear is usually the local or `l` channel. Blank lines then separate each channel block from one another. (An exception is the `defaults` channel, which can appear before the `l` channel).
5. A typical channel definition consists of a channel name (`a_channel`), some keywords which define the configuration of a channel (`defragment charset7 usascii`), and a routing system (`a-daemon`), which is also called a *channel tag*.
6. The contents of other files can 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. Any files included in the configuration file must be world-readable just as the configuration file is world-readable.

The following table shows how some example addresses would be routed by the preceding configuration.

Addresses and Associated Channels

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

Refer to Rewrite Rules, Channel Definitions, and Configuring Rewrite Rules for more information on the MTA configuration file.

Note
Whenever changes are made to the `imta.cnf` file, the MTA configuration must be recompiled. See Compiling the MTA Configuration.

Mappings File

Many components of the MTA employ table lookup-oriented information. This type of table is used to transform, that is *map*, an input string into an output string. Such *mapping tables* are represented as two columns. The first (left-hand) column provides *patterns* to which an input strings is compared. The second (right-hand) column supplies the *template* by which the input string is mapped to an output string. For details about which MTA processes use which tables and when, see Messaging Server Mapping Tables.

Most of the MTA databases that contain different types of MTA data, and which should not be confused with mapping tables, are instances of just this type of table. The MTA database files, however, do not provide wildcard-lookup facilities, owing to inherent inefficiencies in having to scan the entire database for wildcard matches.

The MTA `mappings` file supports multiple mapping tables. Wildcard capabilities are provided, as well as multistep and iterative mapping methods. 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.

Mapping tables are kept in the MTA `mappings` file in the *msg-svr-base*`/config/` directory. The contents of the `mappings` file are incorporated into the compiled configuration as part of the reloadable section (see Compiling the MTA Configuration). Failure to allow world-read access leads to erratic behavior. Whenever changes are made to the `mappings` file, the MTA configuration must be recompiled.

The following table lists the mapping tables.

Messaging Server Mapping Tables

Mapping Table	Description

AUTH_REWRITE	Used with the authrewrite keyword to modify header and envelope addresses using addressing information obtained from authentication operations (SASL). See TCP/IP connection and DNS lookup support in http://msg.wikidoc.info/index.php/Channel_configuration.
CHARSET-CONVERSION	Used to specify what sorts of channel-to-channel character set conversions and message reformatting should be done. See Character Set Conversion and Message Reformatting.
COMMENT_STRINGS	Used to modify address header comments (strings enclosed in parentheses). See how to handle comments in address header lines in http://msg.wikidoc.info/index.php/Channel_configuration.
CONVERSIONS	Used to select message traffic for the conversion channel. See Selecting Traffic for Conversion Processing.
FORWARD	Used to perform forwarding similar to that performed using the alias file or alias database. See The Forward Lookup Table and FORWARD Address Mapping.
FROM_ACCESS	Filter based on envelope From addresses. Used by `tcp_smtp_server` when `MAIL FROM` is received. Use this table if the `To` address is irrelevant. See Controlling Access with Mapping Tables.
INTERNAL_IP	Used to recognize systems and subnets that are internal. Called by entries in the PORT_ACCESS table. See To Add SMTP Relaying.
IP_ACCESS	Block incoming connections based on source channel, IP address count for remote server, index of current IP address being tried. See Controlling Access with Mapping Tables.
MAIL_ACCESS	Filter based on combination of information available in SEND_ACCESS and PORT_ACCESS tables. Used by `tcp_smtp_server` when `RCPT TO` is received. See Controlling Access with Mapping Tables.
NOTIFICATION_LANGUAGE	Used to customize or localize notification messages. See Controlling Delivery Status Notification Messages.
ORIG_MAIL_ACCESS	Same as MAIL_ACCESS except using the value of the *original* To address after rewriting but before alias expansion. See Controlling Access with Mapping Tables.
ORIG_SEND_ACCESS	Same as SEND_ACCESS except using the value of the *original* To address after rewriting but before alias expansion. See Controlling Access with Mapping Tables.
PERSONAL_NAMES	Used to modify personal names (strings preceding angle-bracket-delimited addresses). See handling personal names in address header lines in http://msg.wikidoc.info/index.php/Channel_configuration.
PORT_ACCESS	Block incoming connections based on source and destination IP address and TCP port number. Used by `dispatcher` immediately after accepting the TCP connection. Used by `tcp_smtp_server` when connection is handed off from `dispatcher`. See Controlling Access with Mapping Tables.
REVERSE	Used to convert addresses from an internal form to a public, advertised form. To Convert Addresses from an Internal Form to a Public Form.
SEND_ACCESS	Filter based on envelope From address, envelope To address, and source and destination channels. Used by `tcp_smtp_server` when `RCPT TO` is received. See Controlling Access with Mapping Tables.

*SMS_Channel*_TEXT	Used for site-defined text conversions. See Site-defined Text Conversions.
X-ATT-NAMES	Used to retrieve a parameter value from a mapping table. See To Call Out to a Mapping Table from a Conversion Entry.
X-REWRITE-SMS-ADDRESS	Used for local SMS address validity checks. See Site-defined Address Validity Checks and Translations.

File Format in the Mappings File

The `mappings` 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 using the $ character. 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:

```
<TABLE1_NAME>

   pattern1-1    template1-1
   pattern1-2    template1-2
   pattern1-3    template1-3
      .             .
      .             .
      .             .
   pattern1-n    template1-n

<TABLE2_NAME>

   pattern2-1    template2-1
   pattern2-2    template2-2
   pattern2-3    template2-3
       .            .
       .            .
       .            .
   pattern2-n    template2-n
          .
          .
          .


<TABLE3_NAME>

          .
          .
          .
```

An application using the mapping table `TABLE2_NAME` would map the input string which matched `pattern2-2` into whatever is specified by `template2-2`. Each pattern or template can contain up to 256 and 1024 characters respectively. The maximum size of a line in the mapping file is 4096 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 `mappings` file.

Including Other Files in the Mappings File

Other files may be included in the `mappings` file. This is done with a line of the form:

```
<file-spec
```

This effectively substitutes the contents of the file `file-spec` into the `mappings` 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 `mappings` file. Includes can be nested up to three levels deep. Include files are loaded at the same time the `mappings` file is loaded – when it is compiled. Includes are not loaded on demand, so there is no performance or memory savings involved in using include files.

Mapping Operations

Mapping tables can be thought of like programming subroutines or functions. They take an input string and input flags. They return an output string and output flags. All mapping tables follow the same rules, but the format of the input and output strings, and which input and output flags apply, varies depending on the purpose of the table, which is determined by the MTA process or function which uses the table. For details about which MTA processes use which tables and when, see Messaging Server Mapping Tables.

A mapping operation starts off with an input string (and possibly input flags). 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 input string is compared to the left side, the *pattern*, of the entry in a case-blind fashion. If the input string matches the pattern, the right side, the *template*, is processed. In the simplest form, the mapping table will return the result of processing that template.

The pattern can contain a variety of wildcard characters as well as characters which must be matched exactly.

The template can contain characters which will be output exactly as provided, as well as substitution sequences, which will be replaced with the result of performing another table lookup, or other callout, and metacharacters which can be used to test input flags or other conditions and set output flags. Such tests, call-outs, or lookups can result in success or failure as well as returning an output string or setting output flags. Other metacharacters can specify that if a subsequent test, call-out, or lookup *fails*, the mapping operation should continue scanning down the table for other entries that match the input string rather than returning immediately. Some output flags are simply set or clear. Other output flags require an argument which is passed back to the calling process as the value of that output flag. For more information about the significance of the ordering of output flag arguments, see the sections about the specific mapping tables.

For more information on patterns and templates, see the following subsections:

- Mapping Entry Patterns
- IP Matching
- Mapping Entry Templates

Mapping Entry Patterns

Patterns can contain wildcard characters. In particular, the usual wildcard characters are allowed: an asterisk (*) matches zero or more characters, and each percent sign (%) matches 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.

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.
Glob 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, that is, 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.
$[c]*	Match arbitrary occurrences of character c.

$[c1 c2 ... cn]%	Match exactly one occurrence of character c1, c2, or cn.
$[c1 c2 ... cn]*	Match arbitrary occurrences of any characters c1, c2, or cn.
$[c1 -cn]%	Match any one character in the range c1 to cn.
$[c1 -cn]*	Match arbitrary occurrences of characters in the range c1 to cn.
$< IPv4>	Match an IPv4 address, ignoring bits.
$(IPv4)	Match an IPv4 address, keeping prefix bits.
${IPv6}	Match an IPv6 address. Note that IPv6 connection handling is not currently supported in Messaging Server.

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.

You can use `imsimta test -match` 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 input string. For instance, when the input string `a/b/c` is compared to the pattern `*/*`, the left asterisk matches `a/b` and the right asterisk matches the remainder, `c`.

The `$_` modifier causes wildcard matching to be minimized, where the least possible match is considered the match, working from left to right across the pattern. For instance, when the string `a/b/c` is compared to the pattern `$_*/$_*`, the left `$_*` matches a and the right `$_*` matches `b/c`.

IP Matching

With IPv4 prefix matching, an IP address or subnet is specified, optionally followed by a slash and the number of bits from the prefix that are significant when comparing for a match. For example, the following matches anything in the `123.45.67.0` subnet:

```
$(123.45.67.0/24)
```

With IPv4 ignore bits 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 example, the following matches anything in the `123.45.67.0` subnet:

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

The following example matches anything in the range `123.45.67.4` through `123.45.67.7`:

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

IPv6 matching matches an IPv6 address or subnet.

> Phase 1 of IPv6 support was introduced in **Messaging Server 7 Update 2**.

Mapping Entry Templates

If the input string does not match the pattern in a given entry, no action is taken, and the scan proceeds to the next entry. If the input string matches the pattern, 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 do not appear in the output string produced by a template, but produce some special substitution or processing. See the following table for a list of the special substitution and standard processing metacharacters. Any other metacharacters are listed in the sections about specific mappings tables. See Access Control Mapping Tables.

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.
$]...[	URL lookup; substitute in result.
$|...|	Applies specified mapping table to supplied string.
${...}	General database substitution.
$}*domain*, *attribute*{	Adds the capability to access per-domain attributes. *domain* is the domain in question and *attribute* is the attribute associated with the domain. If the domain exists and has the attribute, its initial value is substituted into the mapping result; if either the attribute or the domain does not exist, the mapping entry fails. *attributes* can be domain LDAP attributes or the special attributes defined below: `_base_dn_` - The base DN for user entries in the domain `_domain_dn_` - The DN of the domain entry itself `_domain_name_` - The name of the domain (as opposed to an alias) `_canonical_name_` - The canonical name associated with the domain
$[...]	Invokes site-supplied routine; substitute in result.
Metacharacter	**Description**

$C	Continues the mapping process starting with the next table entry; uses the output string of this entry as the new input string for the mapping process. See Processing Control ($C, $L, $R, $E).
$E	Ends the mapping process now; uses the output string from this entry as the final result of the mapping process. $+1E exits immediately without interpreting the rest of the template. See Processing Control ($C, $L, $R, $E).
$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 $R metacharacter. See Processing Control ($C, $L, $R, $E).
$R	Continues the mapping process starting with the first entry of the mapping table; uses the output string of this entry as the new input string for the mapping process. See Processing Control ($C, $L, $R, $E).
$nA	Inserts the nth left character of the current address starting from position 0. The entire address is inserted if n is omitted.
$nX	Inserts the nth left component of the mailhost starting from 0. The entire mailhost is inserted if n is omitted.
$?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.
$=	Forces subsequent substituted characters to undergo quoting appropriate for insertion into LDAP search filters.
$:*x*	Match only if the specified flag is set.
$;*x*	Match only if the specified flag is clear.

This section consists of the following subsections:

- Wildcard Field Substitutions ($n)
- Controlling Text Case ($\, $^, $_)
- Processing Control ($C, $L, $R, $E)
- Check for Special Flags ($:x, $;x)
- Entry Randomly Succeeds or Fails ($?x?)
- Sequence Number Substitutions ($#...#)
- URL substitutions, $]...[
- Mapping Table Substitutions ($|...|)
- General Lookup Table or Database Substitutions (${...})
- Expression Substitutions ($`..')
- Site-Supplied Routine Substitutions ($[...])
- Generate UTF-8 Strings

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.siroe.com:

```
PSI$%*::*    $1@$0.psi.siroe.com
```

The input string PSI%1234::USER would also match producing USER@1234.psi.siroe.com 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 control whether and when the mapping process terminates. As described in Mapping Operations, the mapping process normally consists of a single pass, from top to bottom, ending at the first entry with a pattern matching the input string, with the result specified by the template of that entry. Even if the processing of a metacharacter in the template fails, the mapping process would normally terminate because the input string matched the pattern of that entry. The metacharacters $C, $L, and $R cause the mapping process to continue instead of terminate. The metacharacter $E explicitly terminates the mapping process, thus overriding a previous $C on the same line.

The metacharacters $C, $L, and $R can be used to change the input string and continue the mapping process. If the template modifies the output string and $C, $L, or $R are used, the mapping process continues with the output string of the current entry used as the input string for further operations. $C continues with the next entry. $L continues with the next entry, but if no matching entry is found before the end of the table, one more pass will be made starting again at the top of the table. $R continues from the top of the table.

Templates are processed from left to right. When the intention is that an entry should succeed and terminate the mapping process if the template succeeds, but that the mapping process should continue if the template fails, then the entry should use the $C (or $R or $L) metacharacter to the left of the part that may fail and $E to the right of that part.

For example, see the PORT_ACCESS table where the $|...| callout is used in To Use DNS Lookups Including RBL Checking for SMTP Relay Blocking to check the client IP address against the INTERNAL_IP table. If the lookup succeeds, the template processing ends with $Y and the mapping process is terminated by the $E. But if the lookup fails, the template processing ends with the failure, but the $C causes the mapping process to continue. Because this template generated no output string, the original input string remains unmodified.

Because metacharacters $L and $R reiterate the mapping process, 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 new input string that is the same length or longer than the previous pass. If the resulting output string has a shorter length than the input string, the counter is reset to zero. A request to reiterate a mapping is ignored after the counter has exceeded 10.

Note that any of the metacharacters $C, $E, $L, or $R, when present in the template, influences 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.

Check for Special Flags ($:x, $;x)

Some mapping probes have special flags set. You can test their presence or absence using the general mapping table facility of the `$:` or `$;` tests. `$:x` causes an entry to match only if the flag *x* is set. `$;x` causes an entry to match only if the flag *x* is clear. See specific mapping table descriptions Messaging Server Mapping Tables for any special flags that apply for that table.

When the intention is that an entry should succeed and terminate if the flag check succeeds, but that the mapping process should continue if the flag check fails, then the entry should use the `$C` metacharacter to the left of the flag check and use the `$E` flag to the right of the flag check.

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. (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 SMTP email and you'd like to slow it down. You can use a `PORT_ACCESS` mapping table in the following way. Suppose want 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 the MTA 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|*         $N45s$ 4.40$ Try$ again$ later
```

Sequence Number Substitutions ($#...#)

A `$#...#` substitution increments the value stored in an MTA 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>|<m>#
```

```
$#<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 MTA sequence file. 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 is padded with 0s on the left to obtain the correct number of digits. If a width is explicitly specified, then the radix must be explicitly specified also.

The optional *m* argument is a modulus. If this fourth argument is specified, the value inserted is the sequence number retrieved from the file mod *m*. The default is not to perform any modulus operation.

As stated previously, the MTA sequence file referred to in a mapping must already exist. To create an MTA sequence file, use the following UNIX 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 MTA user account (configured to be `nobody` in the `imta_tailor` file) in order to use such sequence number files.

URL substitutions, $]...[

A substitution of the form `$]`*url*`[` is specially handled. *url* can be any supported URL type including `file:` and `data:`. Standard LDAP URLs can also be used, with the host and port omitted. The host and port are instead specified with the `LDAP_HOST` and `LDAP_PORT` options. That is, the LDAP URL should be specified as:

```
ldap:///<dn>[?<attributes>[?<scope>?<filter>]]
```

where the square bracket characters `[` and `]` shown previously indicate optional portions of the URL. The *dn* is required and is a distinguished name specifying the search base. The optional *attributes*, *scope*, and *filter* portions of the URL further refine the information to return. That is, *attributes* specifies the attribute or attributes to be returned from LDAP directory entries matching this LDAP query. The *scope* can be any of `base`, (the default), `one`, or `sub`. *filter* describes the characteristics of matching entries.

Certain LDAP URL substitution sequences are available for use within the LDAP query URL. The length of URLs can be 1024 characters. This also applies to expressions created by mappings and mapping calls to other mappings.

Mapping Table Substitutions ($|...|)

A substitution of the form $|*mapping*;*argument*| is handled specially. The MTA looks for an auxiliary mapping table named *mapping* in the MTA `mappings` file, and uses *argument* as the input to that named auxiliary mapping table. The named auxiliary mapping table must exist and must set the `$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 is used as the output string.

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 means that the processing control metacharacter is not seen.

General Lookup Table or Database Substitutions (${...})

A substitution of the form ${*text*} is handled specially. The *text* part is used as a key to access the general lookup table or database (see MTA Text Databases for more information). If *text* is found in the table, the corresponding template from the table is substituted. If *text* does not match an entry in the table, the input string is used unchanged as the output string.

If you are using the general lookup table you need to set the low order bit of the MTA option `use_text_databases`. That is, set it to an odd number. Changes to the `general.txt` need to be compiled into the MTA configuration using the `imsimta cnbuild` to compile and `imsimta reload` to reload the reloadable data.

If you are using a general database, 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 table substitution, the processing control metacharacter should be placed to the left of the general table substitution in the mapping table template. Otherwise, the "failure" of a general table substitution means that the processing control metacharacter is not seen.

Expression Substitutions ($`..')

A substitution of the form:

```
$`n>>x<<y'
```

where *n* is some value (for example, a $n value from the pattern), and *x* and *y* are the number of bits to shift the value. Consider the following substitution:

```
$`255>>4<<4'
```

This example results in 240; that is, shift right 4 bits causes 1111111111 to become 1111, then shift left 4 bits becomes 11110000.

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 on UNIX, the MTA 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 then replaces 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 is 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 means that the processing control metacharacter is not seen.

The site-supplied routine callout mechanism enables the MTA'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.

Generate UTF-8 Strings

You can generate UTF-8 strings from Unicode character values in the general mapping table facility. A Unicode metacharacter sequence of the form:

```
$&A0A0,20,A1A1&
```

produces a UTF-8 string containing the characters at position `A0A0`, `20`, and `A1A1` in it.

Other MTA Configuration Files

In addition to the `imta.cnf` file, Messaging Server provides several other configuration files to help you configure MTA services. These files are summarized in MTA Configuration Files.

This section consists of the following subsections:

- Alias File
- TCP/IP (SMTP) Channel Option Files
- Conversion File
- Dispatcher Configuration File
- Mappings File
- Option File
- Tailor File
- Job Controller File

If you make changes to the `reverse, forward,` or general databases, then issue the command `imsimta reload` to get the changes to take effect (see MTA Text Databases). If you make changes to the `imta.cnf`, `mappings` file, `aliases`, `conversions`, or `option.dat` files that do not affect the job_controller, then you should issue an `imsimta cnbuild` followed by an `imsimta restart smtp`.

If you make changes to `dispatcher.cnf` you need to do an `imsimta restart dispatcher`. If you make changes to the configuration files that are included in the compiled configuration that affect the job controller, but not the SMTP server, in many cases you should issue the following commands: `imsimta cnbuild` and `imsimta restart job_controller`.

See *Messaging Server Administration Reference* for details on these commands.

MTA Configuration Files

File	Description
Alias File (mandatory)	Implements aliases not present in the directory. *msg-svr-base* `/config/aliases`
TCP/IP (SMTP) Channel Option Files SMTP Option Files)	Sets channel-specific options. *msg-svr-base*`/config/`*channel*`_option`
Conversion File	Used by the conversion channel to control message body part conversions. *msg-svr-base*`/config/conversions`
Dispatcher Configuration File (mandatory)	Configuration file for the Dispatcher. *msg-svr-base* `/config/dispatcher.cnf`
Job Controller File (mandatory)	Configuration file used by the `Job Controller`. *msg-svr-base* `/config/job_controller.cnf`
MTA Configuration File (mandatory)	Used for address rewriting and routing as well as channel definition. *msg-svr-base*`/config/imta.cnf`
Mappings File (mandatory)	Repository of mapping tables./*msg-svr-base*`/config/mappings`
Option File	File of global MTA options. *msg-svr-base*`/config/option.dat`
Tailor File (mandatory)	File to specify locations and some tuning parameters. *msg-svr-base* `/config/imta_tailor`
General Lookup Table (optional)	General lookup facility is equivalent to the general database. Part of reloadable compiled configuration.File to specify locations and some tuning parameters. *msg-svr-base*`/config/general.txt`
Forward Lookup Table (optional)	Lookup facility for To: addresses. Equivalent to forward database. Part of reloadable compiled configuration. *msg-svr-base*`/config/forward.txt`
Reverse Lookup Table (optional)	Reverse lookup facility for From: addresses. Equivalent to reverse database. Part of reloadable compiled configuration *msg-svr-base* `/config/reverse.txt`

Alias File

The alias file, `aliases`, sets aliases not set in the directory. In particular, the address for root is a good example. Aliases set in this file will be ignored if the same aliases exist in the directory. For more information about aliases and the `aliases` file, see Aliases.

After making changes to the `aliases` file, you must restart the MTA for the changes to take effect.

TCP/IP (SMTP) Channel Option Files

TCP/IP channel option files control various characteristics of TCP/IP channels. Channel option files must

be stored in the MTA configuration directory and named *xoption, where _x* is the name of the channel. For example, *msg-svr-base*/config/tcp_local_option. For more information refer to Configuring SMTP Channel Options. For complete information on all channel option keywords and syntax, see *Messaging Server Administration Reference*.

Conversion File

The conversion file, conversions, specifies how the conversion channel performs conversions on messages flowing through the MTA. Any subset of MTA traffic can be selected for conversion and any set of programs or command procedures can be used to perform conversion processing. The MTA looks at the conversion file to choose an appropriate conversion for each body part.

For more information about the syntax of this file, see The Conversion Channel.

Dispatcher Configuration File

The Dispatcher configuration file, dispatcher.cnf, specifies Dispatcher configuration information. 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. (For conceptual information, see The Dispatcher).

The Dispatcher configuration file format is similar to the format of other MTA 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 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=<msg-svr-base>/lib/tcp_smtp_server
LOGFILE=<msg-svr-base>/log/tcp_smtp_server.log
```

For more information about the parameters for this file, see *Messaging Server Administration Reference*.

Mappings File

The `mappings` file defines how the MTA maps input strings to output strings.

Many components of the MTA 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 MTA databases are instances of this type of mapping table. The MTA database files, however, do not provide wildcard-lookup facilities, owing to inherent inefficiencies in having to scan the entire database for wildcard matches.

The `mappings` file provides the MTA with facilities for supporting multiple mapping tables. Full wildcard facilities are provided, and multi-step 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 actually serve to eliminate the need for most of the entries in an equivalent database, and this may actually result in lower overhead overall.

You can test mapping tables with the `imsimta test -mapping` command. For more information about the syntax of the `mappings` file and the `test -mapping` command, see Mappings File and *Messaging Server Administration Reference*.

After making changes to the `mappings` file, you must restart the MTA or issue the command `imsimta reload`.

Option File

The options file, `option.dat`, specifies global MTA options as opposed to channel-specific options.

You can use the options file to override the default values of various parameters that apply to the MTA 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. You can also use the options file to limit the size of messages accepted by the MTA, specify the number of channels allowed in the MTA configuration, set the number of rewrite rules allowed, and so on.

In `option.dat`, lines starting with `#`, `!` or `;` are treated as comment lines, even if the preceding line has a trailing \ meaning that it is being continued. This means that long options that may contain these characters, in particular delivery options, care has to be taken.

For delivery options, where a natural layout would lead to continuation lines starting with a # or !, there is a safe and neat work around.

For more information about the syntax of the options file, see the *Messaging Server Administration Reference*.

Tailor File

The tailor file, `imta_tailor`, sets the location of various MTA components. For the MTA to function properly, the `imta_tailor` file must always reside in the *msg-svr-base*`/config` directory.

Although you can edit this file to reflect the changes in a particular installation, you must do so with caution. After making any changes to this file, you must restart the MTA. Make the changes while the MTA is shut down.

> **Note**
> Do not edit this file unless absolutely necessary.

For complete information on this file, see the *Messaging Server Administration Reference*.

Job Controller File

The Job Controller creates and manages channel jobs for delivering messages. These channel jobs run inside processing pools within the Job Controller. Think of a processing pool as a "place" where the channel jobs are run. The pool provides a computing area where a set of jobs can operate without vying for resources with jobs outside of the pool. For information on Job Controller concepts and channel keyword configuration, see The Job Controller, Processing Pools for Channel Execution Jobs, and Service Job Limits.

The Job Controller file, `job_controller.cnf`, specifies the following channel processing information:

- Defines various pools
- Specifies for all channels, the master program name and the slave program name, if applicable

In the `imta.cnf` file, you can specify the name of a process pool (that was defined in the `job_controller.cnf` file) by using the `pool` *keyword*. For example, the following fragment from a sample `job_controller.cnf` file defines the pool `MY_POOL`:

```
[POOL=MY_POOL]
job_limit = 12
```

The following fragment from a sample `imta.cnf` file specifies the pool `MY_POOL` in a channel block:

```
channel_x pool MY_POOL
channel_x-daemon
```

If you want to modify the parameters associated with the default pool configuration or add additional

pools, you can do so by editing the `job_controller.cnf` file, then stopping and restarting the Job Controller.

The first pool in the Job Controller configuration file is used for any requests that do not specify the name of a pool. The MTA channels defined in the MTA configuration file (`imta.cnf`) can have their processing requests directed to a specific pool by using the `pool` channel keyword followed by the name of the pool. The pool name must match the name of a pool in the Job Controller configuration. If the Job Controller does not recognize the requested pool name, the request is ignored.

In the initial configuration, the following pools are defined: `DEFAULT`, `LOCAL_POOL`, `IMS_POOL`, `SMTP_POOL`

Examples of Use

Typically, you would add additional pool definitions 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 pools 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 pool with the job limit, then use the `pool` channel keyword to direct those channels to the new, more appropriate pool.

In addition to the definition of pools, the Job Controller configuration file also contains a table of the MTA channels and the commands that the Job Controller must use to process requests for each channel. The two types of requests are termed "master" and "slave." Typically, a channel master program is invoked when there is a message stored in an MTA message queue for the channel. The master program dequeues the message.

A slave program is invoked to poll a channel and pick up any messages inbound on that channel. While nearly all MTA channels have a master program, many do not have or need a slave program. For example, a channel that handles SMTP over TCP/IP does not 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 the MTA'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 pool whose job limit is one:

```
[POOL=single_job]
job_limit=1
```

On the other hand, if the destination system has enough parallelism, you can set the job limit to a higher value.

The following example shows a sample Job Controller configuration file. Job Controller Configuration File Options shows the available options.

Sample Job Controller Configuration File in UNIX

```
!MTA Job Controller configuration file
!
!Global defaults
tcp_port=27442          (1)
secret=never mind
slave_command=NULL      (2)
max_life_age=3600       (3)
!
!
!Pool definitions
!
[POOL=DEFAULT]          (4)
job_limit=10            (5)
!
[POOL=LOCAL_POOL]
job_limit=10
!
[POOL=IMS_POOL]
job_limit=1
!
[POOL=SMTP_POOL]
job_limit=1
!
!Channel definitions
!
!
[CHANNEL=l]              (6)
master_command=<msg-svr-base>/lib/l_master
!
[CHANNEL=ims-ms]
master_command=<msg-svr-base>/lib/ims_master
!
[CHANNEL=tcp_*]          (7)
master_command=<msg-svr-base>/lib/tcp_smtp_client
```

The key items in the preceding example (numbered and enclosed in parentheses) are:

1. This global option defines the TCP port number on which the Job Controller listens for requests.
2. Sets a default `SLAVE_COMMAND` for subsequent [`CHANNEL`] sections.
3. Sets a default `MAX_LIFE_AGE` for subsequent [`CHANNEL`] sections.
4. This [`POOL`] section defines a pool named `DEFAULT`.
5. Set the `JOB_LIMIT` for this pool to `10`.
6. This [`CHANNEL`] section applies to a channel named `l`, the UNIX 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_`.

Example of Adding Additional Pools

> **Note**
> The job limit that is set in the `job_controller` is per pool. So, for example, if you have your `SMTP_POOL` defined with a `job_limit` of 10, then only 10 `tcp_smtp` client processes can run in that pool at any given time.

There are situations where you might want to create additional `tcp_*` channels (say, for example, a tcp channel for particularly slow mail sites). It is a good idea to have these channels run in different pools. The reason for this is if you created ten different `tcp_*` channels and they were all running in `SMTP_POOL`, it is possible to only have one `tcp_smtp` client running per `tcp_*` channel at any given time (depending on whether or not there is mail destined for all the `tcp_*` channels and given that `SMTP_POOL` is defined with a `job_limit` of `10`). Assuming there is heavy load on the system and that all queues have messages waiting to go out the various `tcp_*` channels, this would not be efficient. It is much more likely that one would want to define additional pools for the additional `tcp_*` channels so that there is no contention for slots.

For example, suppose you set up the following `tcp_*` channels:

```
tcp_yahoo smtp mx pool yahoo_pool <keyword> <keyword> <keyword>
tcp-yahoo-daemon

tcp_aol smtp mx <keyword> <keyword> <keyword> pool aol_pool
tcp-aol-daemon

tcp_hotmail smtp mx pool hotmail_pool <keyword> <keyword> <keyword>
tcp-hotmail-daemon
...
tcp_sun smtp mx pool sun_pool <keyword> <keyword> <keyword>
tcp-sun-daemon
```

To have ten `tcp_smtp_client` processes for each of the new channels you would add the following in the `job_controller.cnf` file:

```
[POOL=yahoo_pool]
job_limit=10

[POOL=aol_pool]
job_limit=10

[POOL=hotmail_pool]
job_limit=10

 ...

[POOL=sun_pool]
job_limit=10
```

For more information about pools, see Processing Pools for Channel Execution Jobs.

Job Controller Configuration File Options

Option	Description
General Options	*Description*

INTERFACE_ADDRESS=*adapter*	Specifies the IP address interface to which the Job Controller should bind. The value specified (adapter) can be one of ANY, ALL, LOCALHOST, or an IP address. By default the Job Controller binds to all addresses (equivalent to specifying ALL or ANY). Specifying INTERFACE_ADDRESS=LOCALHOST means that the Job Controller only accepts connections from within the local machine. This does not affect normal operation, since no inter-machine operation is supported by the Job Controller. However, this may be inappropriate in an HA environment where an HA agent may be checking if the Job Controller is responding. If the machine on which the Messaging Server is running is in an HA environment, has an "internal network" adapter and an "external network" adapter, and you are not confident of your firewall's ability to block connections to high port numbers, consider specifying the IP address of the "internal network" adapter.
MAX_MESSAGES=*integer*	The Job Controller keeps information about messages in an in-memory structure. In the event that a large backlog builds, it may need to limit the size of this structure. If the number of messages in the backlog exceeds the parameter specified here, information about subsequent messages is not kept in memory. Mail messages are not lost because they are always written to disk, but they are not considered for delivery until the number of messages known by the Job Controller drops to half this number. At this point, the Job Controller scans the queue directory mimicking an `imsimta cache -sync` command. The minimum value is 10. See The Job Controller for more information. The default is 100000.
SECRET=*file_spec*	Shared secret used to protect requests sent to the Job Controller.
SYNCH_TIME=*time_spec*	The Job Controller occasionally scans the queue files on disk to check for missing files. By default, this takes place every four hours, starting four hours after the Job Controller is started. The format of the *time_spec* is *HH*:*MM*/*hh*:*mm* or /*hh*:*mm*. The variable *hh*.*mm* is the interval between the events in hours (*h*) and minutes (*m*). The variable *HH*:*MM* is the first time in a day the even should take place. For example specifying, 15:45/7:15 starts the event at 15:45 and every seven hours and fifteen minutes from then.
TCP_PORT=*integer*	Specifies the TCP port on which the Job Controller should listen for request packets. Do not change this unless the default conflicts with another TCP application on your system. If you do change this option, change the corresponding IMTA_JBC_SERVICE option in the MTA tailor file, *msg-svr-base*/config/imta_tailor, so that it matches. The TCP_PORT option applies globally and is ignored if it appears in a [CHANNEL] or [POOL] section.
Pool Option	*Description*
JOB_LIMIT=*integer*	Specifies the maximum number of processes that the pool can use simultaneously (in parallel). The JOB_LIMIT applies to each pool individually. The maximum total number of jobs is the sum of the JOB_LIMIT parameters for all pools. If set outside of a section, it is used as the default by any [POOL] section that does not specify JOB_LIMIT. This option is ignored inside of a [CHANNEL] section.
Channel Option	*Description*
MASTER_COMMAND=*file_spec*	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 [POOL] section.

MAX_LIFE_AGE= *integer*	Specifies the maximum life time for a channel master job in seconds. If this parameter is not specified for a channel, then the global default value is used. If no default value is specified, 14400 (240 minutes) is used.
MAX_LIFE_CONNS= *integer*	In addition to the maximum life age parameter, the life expectancy of a channel master job is limited by the number of times it can ask the Job Controller if there are any messages. If this parameter is not specified for a channel, then the global default value is used. If no default value is specified, 300 is used.
SLAVE_COMMAND= *file_spec*	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. Most MTA 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 does not specify a SLAVE_COMMAND. This option is ignored inside of a [POOL] section.

Aliases

The MTA 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. For a discussion on how alias resolution is handled see The $V Metacharacter.

Old-style mailing lists defined in the `aliases` file or aliases database now accept a nonpositional `[capture]` parameter. If used, the `[capture]` parameter specifies a capture address with the same semantics as capture addresses specified by the `LDAP_CAPTURE` attribute applied to a user or group in LDAP.

A value "\" given as an `[envelope_from]` nonpositional alias parameter, as an error to positional alias parameter, or as a value of the `mgrpErrorsTo` LDAP attribute, is now interpreted as a request to revert to using the original envelope `From:` address for the incoming message while retaining mailing list semantics. This can be useful for setting up mailing lists that report all forms of list errors to the original sender.

The Alias Database

Use of the Alias Database is discouraged. Use the `aliases` file instead, especially since it can be dynamically reloaded using the `imsimta reload` command.

The MTA 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.

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

The Alias File

The `aliases` 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. Changes can be activated by issuing the `imsimta reload` command (or restarting the MTA). Any line that

begins with an exclamation point is considered to be a comment and is ignored. Blank lines are also ignored.

> **Note**
> Messaging Server provides other facilities for address manipulation, such as the address reversal database and specialized mapping tables. For best performance, however, rewrite rules should be used whenever possible to perform address manipulations. See Configuring Rewrite Rules.

A physical line in this file is limited to 1024 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>@<domain>: <address> (for users in hosted domains)

<user>@<domain>: <address> (for users in non-hosted domains. Example:
default-domain)
```

For example:

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

! A message store user
ms_testuser@siroe.com: mstestuser@ims-ms-daemon
```

Including Other Files in the Alias File

Other files can be included in the primary `aliases` file. A line of the following form directs the MTA 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 `aliases` file. For example, it must be world readable.

The contents of the included file are inserted into the `aliases` 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 `aliases` file itself. Indeed, include files may themselves include other files. Up to three levels of include file nesting are allowed.

Command-Line Utilities

Messaging Server provides several command-line utilities that enable you to perform various maintenance, testing, and management tasks for the MTA. For example, you use the `imsimta cnbuild` command to compile the MTA configuration, alias, mapping, security, system wide filter, and option files. For complete information on the MTA command-line utilities, see the *Messaging Server Administration Reference*.

SMTP Security and Access Control

For information about SMTP security and access control, see Mail Filtering and Access Control.

Log Files

All MTA specific log files are kept in the log directory, (*msg-svr-base*`/log`). This directory contains log files that describe message traffic through the MTA and log files that describe information about specific master or slave programs.

For more information about MTA log files, see Managing Logging.

To Convert Addresses from an Internal Form to a Public Form

Addresses can be converted from an internal form to a public, advertised form using the Address-Reversal text database (also called the *reverse text database*) and the REVERSE mapping table. For example, while `uid@mailhost.siroe.com` might be a valid address within the `siroe.com` domain, it might not be an appropriate address to expose to the outside world. You might prefer a public address like `firstname.lastname@siroe.com`.

Messaging Server provides other facilities for address manipulation, such as the `aliases` file and specialized mapping tables. For best performance, however, rewrite rules should be used whenever possible to perform address manipulations. See Configuring Rewrite Rules.

This section consists of the following subsections:

- MTA Text Databases
- To Set Address Reversal Controls
- The Forward Lookup Table and FORWARD Address Mapping

In the reverse text database, the public address for each user is specified by the `mail` attribute of the user entry in the directory.

The reverse text database contains a mapping between a valid address and a public address. See MTA Text Databases for more information.

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 `mappings` file. No substitution is made, and rewriting terminates normally if the table does not exist or no entries from the table match.

If a REVERSE mapping table is found in the `mappings` file, and if the address matches a mapping entry, the resulting string replaces the address if the entry specifies a `$Y`. A `$N` discards the result of the mapping. If the mapping entry specifies `$D` in addition to `$Y`, the resulting string runs through the reversal database once more; and if a match occurs, the template from the database replaces the mapping result (and hence the address). The form of general REVERSE mapping table entries (that is, entries that apply to all channels) is shown below. Note that flags can be either in front of the new address or at the end.

```
REVERSE

   OldAddress        $Y[Flags]NewAddress
```

The following shows the form of *channel-specific* entries (that is, mapping that only occurs only on messages passing through a specific channel). You must set `use_reverse_database` to 13 in the `option.dat` for channel-specific entries to work.

```
REVERSE

   source-channel|destination-channel|OldAddress  $Y[Flags]NewAddressS
```

The following table shows REVERSE mapping table flags.

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.

MTA Text Databases

MTA use of sleepycat databases is being deprecated because of the instability it introduces in Messaging Server deployments. (Note that sleepycat will not be removed in the near future). As a result, MTA text databases for the reverse, forward and general databases should be used instead.

To set up text databases:

1. Prepare a text file containing the data.
   This is in the same format that `imsimta crdb` uses: one entry per line with two fields separated by one or more spaces. The file names are specified by the `IMTA_GENERAL_DATA`, `IMTA_REVERSE_DATA`, and `IMTA_FORWARD_DATA` options in `imta_tailor`), which normally point, respectively, to `IMTA_TABLE:general.txt`, `IMTA_TABLE:reverse.txt`, and `IMTA_TABLE:forward.txt` in *msg-svr-base*`/config/`.
   `general.txt` - general database
   `reverse.txt` - reverse database
   `forward.txt` - forward database
2. Set the appropriate bit or bits in the USE_TEXT_DATABASES option:
   bit 0 (value 1) - use text file for general databasebit 1 (value 2) - use text file for reverse database
   bit 2 (value 4) - use text file for forward database

3. Set whatever additional options are needed to enable the desired databases.
   For example, USE_REVERSE_DATABASE, USE_FORWARD_DATABASE, or whatever
4. Run `imsimta cnbuild`.
5. Run `imsimta reload`.

The only case where USE_TEXT_DATABASES is not appropriate is for highly dynamic data. In those cases, you are better served by writing your own MTA plug-ins rather than by relying on the built-in database support.

If the text database is not appropriate, and you want to use the `crdb` (Sleepycat) database support, then it may be possible, by structuring your database usage style and updating process appropriately, to use either `imsimta crdb` or `imsimta db` to update the database without recompiling, reloading, or restarting. However, for this to work you either have to be in a situation where you can only add or update existing entries, in which case you can use `imsimta crdb`. Otherwise, you have to have your data structured as a series of add/delete/change operations. If your data isn't structured this way (and it usually is not), you're back to replacing the entire database when updating, which in this case, makes text databases preferable.

To Set Address Reversal Controls

The `reverse` and `noreverse` channel keywords, and the MTA options USE_REVERSE_DATABASE and REVERSE_ENVELOPE are used to control the specifics of when and how address reversal is applied. By default, the address reversal operation applies to all addresses, not just to backward pointing addresses.

Address reversal can be enabled or disabled by setting the value of the REVERSE_ENVELOPE system option (Default: 1-on, 0-off).

`noreverse` on the destination channel specifies that address reversal is not applied to addresses in messages. `reverse` specifies that address reversal is applied. See Enabling Channel-Specific Use of the Reverse Database for details.

USE_REVERSE_DATABASE controls whether the MTA uses the address reversal text database and REVERSE mapping as a source of substitution addresses. A value of 0 means address reversal is not used with any channel. A value of 5 (default) specifies that address reversal is applied to all addresses, not just to backward pointing addresses, after they have been rewritten by the MTA address rewriting process. A value of 13 specifies that address reversal is applied to addresses with the `reverse` channel keyword, not just to backward pointing addresses, after they have been rewritten by the MTA address rewriting process. Further granularity of address reversal operation can be specified by setting the bit values of the USE_REVERSE_DATABASE option.

The REVERSE_ENVELOPE option controls whether or not address reversal is applied to envelope `From` addresses as well as message header addresses.

See the detailed descriptions of these options and keywords in *Messaging Server Administration Reference* for additional information on their effects.

General Reverse Mapping Example

An example of a general REVERSE Mapping is as follows: suppose that the internal addresses at `siroe.com` are of the form `user@mailhost.siroe.com`. However, the user name space is such that `user@host1.siroe.com` and `user@host2.siroe.com` specify the same person for all hosts at `siroe.com`. The following REVERSE mapping may be used in conjunction with the address-reversal text database:

```
REVERSE

   *@*.siroe.com        $0@siroe.com$Y$D
```

In this example, addresses of the form *name*@*anyhost*`.siroe.com` would be changed to *name* `@siroe.com.` The `$D` metacharacter causes the address-reversal database to be consulted. The address-reversal text database should contain entries of the form:

```
user@mailhost.siroe.com     first.last@siroe.com
```

Channel-Specific Reverse Mapping Example

By default, the address reversal text database is used if the routability scope is set to the mail server domains. An example of a channel-specific REVERSE mapping table entry would be as follows:

```
REVERSE

   tcp_*|tcp_local|binky@macho.siroe.com    $D$YRebecca.Woods@siroe.com
```

This entry tells the MTA that for any mail with source channel of `tcp_*` going out the destination channel of `tcp_local` to change addresses of the form `binky@macho.siroe.com` to `Rebecca.Woods@siroe.com`

> **Note**
> To enable channel-specific reverse mapping, you must set USE_REVERSE_DATABASE option in `option.dat` to 13. (Default=5.)

The Forward Lookup Table and FORWARD Address Mapping

Address reversals are not applied to envelope To: addresses. The reasons for this omission are fairly obvious: envelope To: addresses are continuously rewritten and modified as messages proceed through the mail system. The entire goal of routing is to convert envelope To: addresses to increasingly system and mailbox-specific formats. The canonicalization functions of address reversal are entirely inappropriate for envelope To: addresses.

In any case, plenty of machinery is available in the MTA to perform substitutions on envelope To: addresses. The alias file, alias database and general lookup table, provide precisely this functionality.

The MTA also provides the forward lookup table and FORWARD mapping, used for special sorts of forwarding purposes, such as pattern-based forwarding, source-specific forwarding, or auto-registration of addresses. Note that the forward lookup table and FORWARD mapping are intended for use primarily for certain special sorts of address forwarding; most sorts of address forwarding, however, are better performed using one of the MTA's other forwarding mechanisms.

The various substitution mechanisms for envelope To: addresses provide functionality equivalent to the

reversal lookup table, but none yet discussed provide functionality equivalent to the reverse mapping. And circumstances do arise where mapping functionality for envelope To: addresses is useful and desirable.

The FORWARD Mapping Table

The FORWARD mapping table provides this functionality of pattern based forwarding, and also provides a mechanism for source specific forwarding. 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. See the following table for a list of additional flags.

FORWARD Output Mapping Table Flags Description

Flag	Description
$D	Run output through the rewriting process again
$G	Run output through the forward lookup table, if forward lookup table use has been enabled
$H	Disable further forward lookup table or FORWARD mapping lookups
$I	Hold the message as a .HELD file
$N	Address remains unchanged
$Y	Use output as new address

The FORWARD mapping, if present, is consulted before any forward lookup table lookup. If a FORWARD mapping matches and has the flag $G, then the result of the FORWARD mapping will be checked against the forward lookup table, if forward lookup table use has been enabled via the appropriate setting of USE_FORWARD_DATABASE. (If channel specific forward lookup table use has been specified, then the source address and source channel will be prefixed to the result of the FORWARD mapping before looking up in the forward lookup table.) If a matching FORWARD mapping entry specifies $D, then the result of the FORWARD mapping (and optional forward table lookup) will be run through the MTA address rewriting process again. If a matching FORWARD mapping entry specifies $H, then no further FORWARD mapping or database lookups will be performed during that subsequent address rewriting (that resulting from the use of $D).

The following input flags are now available in the FORWARD mapping. Previously they were only available to the various *_ACCESS mappings.

FORWARD Input Mapping Table Flags Description

Flag	Description
$A	SASL used to authenticate connection.
$D	NOTIFY=DELAYS active for this recipient.
$E	Incoming connection used ESMTP/EHLO.
$F	NOTIFY=FAILURES active for this recipient.
$L	Incoming connection used LMTP/LHLO.
$S	NOTIFY=SUCCESSES active for this recipient.
$T	SSL/TLS used to secure connection.

The following example illustrates the use of a complex REVERSE and FORWARD mapping. Suppose that a system or pseudo domain named am.sigurd.innosoft.com associated with the mr_local channel produces RFC 822 addresses of the general form:

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

or

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

Although these addresses are perfectly legal they often confuse other mailers that do not fully comply with RFC 822 syntax rules, for example, mailers that do not handle quoted addresses properly. Consequently, an address format that does not require quoting tends to operate with more mailers. One such format is:

```
firstname.lastname@am.sigurd.example.com
```

Example of a complex FORWARD and REVERSE mapping:

```
REVERSE

 *|mr_local|"*,$ *"@am.sigurd.example.com $Y"$1,$ $2"@am.sigurd.example.com
 *|mr_local|"*,*"@am.sigurd.example.com   $Y"$1,$ $2"@am.sigurd.example.com
 *|*|"*,$ *"@am.sigurd.example.com        $Y$3.$2@am.sigurd.example.com
 *|*|"*,*"@am.sigurd.example.com          $Y$3.$2@am.sigurd.example.com
 *|mr_local|*.*@am.sigurd.example.com     $Y"$2,$ $1"@am.sigurd.example.com
 *|*|*.*@am.sigurd.example.com            $Y$2.$3@am.sigurd.example.com

FORWARD

 "*,$ *"@am.sigurd.example.com            $Y"$0,$ $1"@am.sigurd.example.com
 "*,*"@am.sigurd.example.com              $Y"$0,$ $1"@am.sigurd.example.com
 *.*@am.sigurd.example.com                $Y"$1,$ $0"@am.sigurd.example.com
```

So the goals of the sample mapping tables in this example are threefold. (1) Allow any of these three address formats above to be used. (2) Present only addresses in the original format to the mr_local channel, converting formats as necessary. (3) Present only addresses in the new unquoted format to all other channels, converting formats as necessary. (The REVERSE mapping shown assumes that bit 3 in the MTA option USE_REVERSE_DATABASE is set.

The Forward Lookup Table

In cases where address forwardings need to be auto-registered or source specific, the forward lookup table is available. Use of the Forward lookup table for simple forwarding of messages is generally not appropriate. The `aliases` file or alias lookup table is a more efficient way to perform such forwarding. By default, the forward lookup table is not used at all. Its use must be explicitly enabled by using the `USE_FORWARD_DATABASE` option. Forward table lookups are performed after address rewriting and after alias expansion is performed, and after any `FORWARD` mapping is checked. If a forward table lookup succeeds, the resulting substituted address is then run through the MTA address rewriting process all over again.

There are two mechanisms available for the forward lookup table, an in-memory hash table or conventional text database. Unless the size of the table is prohibitively large then hash table is recommended. (1,000 is not prohibitively large, 100,000 is). The hash table is enabled by setting bit 2 (value 4) in the `use_text_databaseS` option as well as setting `use_forward_database`. The hash table is read from the *msg-svr-base*`/configure/forward.txt` file, compiled into the reloadable part of the configuration, and can be forced to be reloaded into the active MTA processes by the `imsimta reload` command.

The format of the source text file by default is expected to be:

```
user1@domain1 changedmailbox1@changeddomain1
user2@domain2 changedmailbox@changeddomain2
```

But if source specific use of the forward text database has been enabled by setting bit 2 of the `USE_FORWARD_DATABASE` option, then the source text file format expected is:

```
source-channel|source-address|original-address changed-address
```

For instance, an entry such as

```
tcp_limited|bob@blue.com|helen@red.com  "helen of troy"@siroe.com
```

will map the To: address helen@red.com to "helen of troy"@siroe.com if and only if the message is coming from bob@blue.com and the enqueuing channel is `tcp_limited.`

See MTA Text Databases for more information on the Forward Text database.

Controlling Delivery Status Notification Messages

Delivery status notifications or *status notifications* are email status messages sent by the MTA to the sender and, optionally, the postmaster. Messaging Server enables you to customize notification messages by content and language. You can also create different messages for each type of delivery status (for example, `FAILED`, `BOUNCED`, `TIMEDOUT`, and so on.). In addition, you can create status notifications for messages originating from specific channels.

By default, status notifications are stored in the *msg-svr-base*`/config/locale/C` directory (specified by the `IMTA_LANG` setting in the *msg-svr-base*`/config/imta_tailor` file). The filenames are as follows:

`return_bounced.txt`, `return_delivered.txt` `return_header.opt`, `return_timedout.txt`, `return_deferred.txt`, `return_failed.txt`, `return_prefix.txt`, `return_delayed.txt`, `return_forwarded.txt`, `return_suffix.txt`

Message text for `*.txt` files should be limited to 78 characters per line. You should not change these

files directly since they will be overwritten when the current version of Messaging Server is upgraded. If you want to modify these files and use them as your only set of notification message template files (`return_*.txt`), copy the files to a new directory and edit them there. Then, set the `IMTA_LANG` option in the `imta_tailor` file to point to the new directory containing those templates. If you want to have multiple sets of notification files (for example, a set for each language) then you will need to set up a `NOTIFICATION_LANGUAGE` mapping table.

This section consists of the following subsections:

To Construct and Modify Status Notifications

A single notification message is constructed from a set of three files: `return_prefix.txt + return_`*ActionStatus*`.txt + return_suffix.txt`

To customize or localize notifications, you would create a complete set of `return_*.txt` files for each locale and/or customization and store it in a separate directory. For example, you could have French notification files stored in one directory, Spanish for another, and notifications for a special unsolicited bulk email channel stored in a third.

> **Note**
> Sample files for French, German, and Spanish are included in this release. These files can be modified to suit your specific needs.
>
> For double-byte languages such as Japanese, be sure to construct your text in Japanese, then view the text as if it was ASCII to check for % characters. If there are accidental % characters, then replace them with %%.

The following information describes the format and structure of status notification message sets.

1. `return_prefix.txt` provides appropriate header text as well as introductory material for the body. The default for US-english locale is as follows:

```
Content-type: text/plain; charset=us-asci
Content-language: EN-US

This report relates to a message you sent with the following
header fields: %H
```

   Non-US-ASCII status notification messages should change the `charset` parameter and `Content-Language` header value appropriately (for example, for French localized files the values would be `ISO-8859-1` and `fr`). `%H` is a header substitution sequence defined in Notification Message Substitution Sequences.

1. `return_`*ActionStatus*`>.txt` contains status-specific text. *ActionStatus* refers to a message's MTA status type. For example, the default text for `return_failed.txt` is as follows:
   `Your message cannot be delivered to the following recipients:%R`
   The default text for `return_bounced.txt` is:
   `Your message is being returned. It was forced to return bythe`

```
postmaster.
The recipient list for this message was:%R
```

2. `return_suffix.txt` contains concluding text. By default this file is blank.

Notification Message Substitution Sequences

Substitutions	Definition
%H	Expands into the message's headers.
%C	Expands into the number of units [1] the message has been queued.
%L	Expands into the number of units [1] the message has left in the queue before it is returned.
%F	Expands into the number of units [1] a message is allowed to stay in the queue.
%S [%s]	Expands to the letter S or s if the previously expanded numeric value was not equal to one. Example: "%C day%s" can expand to "1 day" or "2 days" depending on how many days the message has been queued.
%U [%u]	Expands into the time units Hour [hour] or Day [day], in use. Example: "%C %U%s" can expand to "2 days" or "1 hour" depending on how many days or hours the message has been queued, and the value of the MTA option `RETURN_UNITS`. If you have set `RETURN_UNITS=1` (hours) and your site is using localized status notification messages, you will need to edit `return_delayed.txt` and `return_timedout.txt` and replace the word "days" with the word hours for all languages other than English. French, replace jour(s) with heure(s). German, replace Tag(e) with Stunde(n). Spanish, replace día(s) with hora(s).
%R	Expands into the list of the message's recipients.
%%	% (The text is scanned byte by byte for substitutions sequences regardless of character set. Check for unintended % signs if you are using a double byte character set.)
[1] Units is defined by the RETURN_UNITS option in the MTA Options file and can be hours or days (default).	

To Customize and Localize Delivery Status Notification Messages

Delivery Status Notification Messages can be localized such that messages will be returned to different users in different languages. For example, French notifications could be returned to users who have expressed a preference for French.

Localizing or customizing status notification messages consists of two steps:

1. Create a set of localized/customized `return_*.txt` message files and store each set in a separate directory. This is described in To Construct and Modify Status Notifications.
2. Set up a `NOTIFICATION_LANGUAGE` mapping table.

The `NOTIFICATION_LANGUAGE` mapping table (located in *msg-svr-base*`/config/mappings`) specifies the set of localized or customized notification message files to use depending upon attributes (for example: language, country, domain, or address) of the *originating message* (the message causing the

notification to be sent).

The original sender's message is parsed to determine status notification type, source channel, preferred language, return address and first recipient. Depending on how the table is constructed, a set of notification files is selected depending on one or more of these attributes.

The format of the NOTIFICATION_LANGUAGE mapping table is as follows. The sample entry line has been wrapped for typographic reasons. The actual entry should appear on one physical line.

```
NOTIFICATION_LANGUAGE

 dsn-type-list|source-channel|preferred-language|return-address \
|first-recipient $Idirectory-spec
```

- dsn-type-list is a comma-separated list of delivery status notification types. If multiple types are specified, they must be separated by commas and without spaces (a space will terminate the pattern of the mapping table entry). The types are as follows:
  - failed - Generic permanent failure messages (for example, no such user). Uses the return_failed.txt file.
  - bounced - Notification message used in conjunction with manual "bounces." Done by the postmaster. Uses the return_bounced.txt file.
  - timedout - The MTA has been unable to deliver the message within the time allowed for delivery. The message is now being returned. Uses the return_timedout.txt file.
  - delayed - The MTA has been unable to deliver the message, but will continue to try to deliver it. Uses the return_delayed.txt file.
  - deferred - Non-delivery notification similar to "delayed" but without an indication of how much longer the MTA will continue to try to deliver. Uses the return_deferred.txt file.
  - forwarded - A delivery receipt was requested for this message, however, this message has now been forwarded to a system that does not support such receipts. Uses the return_forwarded.txt file.
- source-channel is the channel generating the notification message, that is, the channel at which the message is currently queued. For example, ims-ms for the message store's delivery queue, tcp_local for outbound SMTP queues, etc.
- preferred-language is the language expressed in the message being processed (the message for which the notification is being generated). The sources for this information are first the accept_language field. If that does not exist, the Preferred-language: header field and X-Accept-Language: header field are used. For a list of standard language code values, refer to the file *msg-svr-base*/config/languages.txt.
  This field, if not empty, will be whatever the originator of the message specified for the Preferred-language: or X-Accept-language: header line. As such, you may find nonsense characters in this field.
- return-address is the envelope From: *address* of the originating message. This is the envelope address to which the notification message will be sent and hence a possible indicator of what language to use.
- first-recipient is the envelope To: address (the first one, if the message is failing to more than one recipient) to which the original message was addressed. For example, in the notification, dan@siroe.com is the envelope To: address being reported on: "your message to dan@siroe.com could not be delivered"
- directory-spec is the directory containing the return_*.txt files to use if the mapping table probe matches. Note that a $I must precede the directory specification.
  For instance, a site that stores French notification files (return_*.txt) in a directory /lc_messages/table/notify_french/ and Spanish notification files in return_*.txt files in a directory /lc_messages/table/notify_spanish/ might use a table similar to the following one. Each entry must start with one or more spaces, and that there can be no blank lines between entries.

```
NOTIFICATION_LANGUAGE

! Preferred-language: header value specified
!
    *|*|fr|*|*     $I/lc_messages/table/notify_french/
    *|*|es|*|*     $IIMTA_TABLE/notify_spanish/
    *|*|en|*|*     $I/imta/lang/
!
! If no Preferred-language value, then select notification based on the
! country code in the domain name. EX: PF=French Polynesia; BO=Bolivia
!
    *|*|*|*.fr|*   $I/imta/table/notify_french/
    *|*|*|*.fx|*   $I/imta/table/notify_french/
    *|*|*|*.pf|*   $I/imta/table/notify_french/
    *|*|*|*.tf|*   $I/imta/table/notify_french/
    *|*|*|*.ar|*   $I/imta/table/notify_spanish/
    *|*|*|*.bo|*   $I/imta/table/notify_spanish/
    *|*|*|*.cl|*   $I/imta/table/notify_spanish/
    *|*|*|*.co|*   $I/imta/table/notify_spanish/
    *|*|*|*.cr|*   $I/imta/table/notify_spanish/
    *|*|*|*.cu|*   $I/imta/table/notify_spanish/
    *|*|*|*.ec|*   $I/imta/table/notify_spanish/
    *|*|*|*.es|*   $I/imta/table/notify_spanish/
    *|*|*|*.gp|*   $I/imta/table/notify_spanish/
    *|*|*|*.gt|*   $I/imta/table/notify_spanish/
    *|*|*|*.gy|*   $I/imta/table/notify_spanish/
    *|*|*|*.mx|*   $I/imta/table/notify_spanish/
    *|*|*|*.ni|*   $I/imta/table/notify_spanish/
    *|*|*|*.pa|*   $I/imta/table/notify_spanish/
    *|*|*|*.ve|*   $I/imta/table/notify_spanish/
```

Note
A default `mappings.locale` file is provided with the installation and will be included in the `mappings` file to enable notification language mapping. To disable notification language mapping, comment out the include line as follows:

```
! <IMTA_TABLE:mappings.locale
```

(Read the comments in the file and modify to suit your needs.)

Internationalization of Generated Notices

Two option files can be used for both the delivery status and message disposition notification. These are intended to make internationalization of generated notices more flexible. They are as follows:

```
IMTA_LANG:return_option.opt (DSN)IMTA_LANG:disposition_option.opt (MDN)
```

The following table describes the available options for these files.

Delivery Status and Message Disposition Notification Options

Option	Description
DAY (DSN)	The text to insert for a %U or %u substitution when RETURN_UNITS=0 (the default) is set. Note that no distinction is made between %U and %u (unlike the default case where English “Day” or “day”, respectively, would be substituted).
DIAGNOSTIC_CODE (DSN)	Override for the “Diagnostic code:“ text used in the construction of the per-recipient section of the first part of a DSN. This field should be specified in the same charset that’s used for the first part of the DSN.
HOUR (DSN)	The text to insert for a %U or %u substitution when RETURN_UNITS=1 is set. Note that no distinction is made between %U and %u (unlike the default case where English “Hour” or “hour”, respectively, would be substituted).
n.n.n (DSN)	When constructing the per-recipient part of a DSN a check will be made to see if there’s an option whose name matches the numeric per-recipient status. If there is the corresponding text will be inserted into the DSN. Additionally, if the REASON option specified above produces a zero length result the REASON field will not be inserted.
ORIGINAL_ADDRESS (DSN)	Override for the “Original address:” text used in the construction of the per-recipient section of the first part of a DSN. This field should be specified in the same charset that’s used for the first part of the DSN.
REASON (DSN)	Override for the “Reason:” text used in the construction of the per-recipient section of the first part of a DSN. This field should be specified in the same charset that’s used for the first part of the DSN.
RECIPIENT_ADDRESS (DSN)	Override for the “Recipient address:” text used in the construction of the per-recipient section of the first part of a DSN. This field should be specified in the same charset that’s used for the first part of the DSN.
RETURN_PERSONAL (DSN and MDN)	Override personal name field to be used in conjunction with the From: field. This field should be RFC 2047 encoded. The value set by the RETURN_PERSONAL MTA option is used if this option is not specified.
SUBJECT (DSN and MDN)	Override Subject: field. This value will only be used if the notification didn’t provide a subject field of its own. This field should be RFC 2047 encoded. An appropriate subject will be constructed if this option is not used and the notification did not provide one.
TEXT_CHARSET (MDN)	Charset text for the first part and subject of the MDN should be converted to. The default is not to perform any conversion.

Additional Status Notification Message Features

The essential procedures for setting up status notification messages is describe in the previous sections. The following sections describe additional functionality:

To Block Content Return on Large Messages

Typically, when a message is bounced or blocked, the content of the message is returned to sender and to the local domain postmaster in the notification message. This can be a strain on resources if a number of very large messages are returned in their entirety. To block the return of content for messages over a certain size, set the CONTENT_RETURN_BLOCK_LIMIT option in the MTA options file.

The MTA fetches the block limit associated with the envelope return address and will set RET=HDRS if no return policy is specified and the message size exceeds the block limit. This prevents nondelivery reports

for large messages from being undeliverable themselves. No new options or settings are associated with this change.

To Remove non-US-ASCII Characters from Included Headers in the Status Notification Messages

The raw format for Internet message headers does not permit non-US-ASCII characters. If non-US-ASCII characters are used in a message header they are encoded using "MIME header encoding" described in RFC 2047. Thus, a Chinese "Subject" line in an email message looks like the following:

```
Subject: =?big5?Q?=A4j=AB=AC=A8=B1=AD=B1=B0=D3=F5=A5X=AF=B2?=
```

and it is the responsibility of email clients to remove the encoding when displaying headers.

Because the %H template copies headers into the body of the notification message, the encoded header text will normally appear. However, Messaging Server will remove the encoding if the character set in the subject (in this case "big5") matches the character set in the `Content-Type` header character set parameter in `return_prefix.txt`. If it does not match, the Messaging Server will leave the encoding intact.

To Set Notification Message Delivery Intervals

Keywords: `notices`, `nonurgentnotices`, `normalnotices`, `urgentnotices`

Undeliverable messages are held in a given channel queue for specified amount of time before being returned to sender. In addition, a series of status/warning messages can be returned to the sender while Messaging Server attempts delivery. The amount of time and intervals between messages can be specified with the `notices`, `nonurgentnotices`, `normalnotices`, or `urgentnotices` keywords. Examples:

```
notices 1 2 3
```

Transient failure status notification messages are sent after 1 and 2 days for all messages. If the message is still not delivered after 3 days, it is returned to its originator.

```
urgentnotices 2,4,6,8
```

Transient failure notifications are sent after 2, 4, and 6 days for messages of urgent priority. If the message is still not delivered after 8 days, it is returned to its originator.

The `RETURN_UNITS` option in the MTA Options file enables you to specify the units in either hours (1) or days (0). The default is days (0). If you set `RETURN_UNITS=1`, then you need to schedule the return job to run hourly as well to get hourly notices. When the return job runs every hour it will also roll over the `mail.log*` files every hour. To prevent the hourly rollover of the `mail.log` file, set the `IMTA_RETURN_SPLIT_PERIOD` tailor file option in the `imta.tailor` file to 24. Return job scheduling is controlled by the `local.schedule.return_job` `configutil` parameter. However, by default this command is run on a regular basis (see Pre-defined Automatic Tasks).

If no `notices` keyword is specified, the default is to use the `notices` setting for the local, `l`, channel. If no setting has been made for the local channel, then the default is to use `notices 3, 6, 9, 12`.

To Include Altered Addresses in Status Notification Messages

Keywords: `includefinal`, `suppressfinal`, `useintermediate`

When the MTA 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 the MTA. The MTA 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 the MTA also includes the final form of the address. Suppressing the inclusion of the final form of the 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 status notification messages. `includefinal` is the default and includes the final form of the recipient address. `suppressfinal` causes the MTA to suppress the final address form, if an original address form is present, from status notification messages.

The `useintermediate` keyword uses an intermediate form of the address produced after list expansion, but prior to user mailbox name generation. If this information isn't available, the final form is used.

To Send, Block and Specify Status Notification Messages to the Postmaster

By default, a copy of failure and warning status notification messages are sent to the postmaster unless error returns and warnings are completely suppressed with a blank `Errors-to:` header line or a blank envelope `From:` address. Further granularity of notification message delivery to the postmaster can be controlled by a number of channel keywords described in the following sections and in Keywords for Sending Notification Messages to the Postmaster and Sender.

Returned Failed Messages

Keywords: `sendpost`, `nosendpost`, `copysendpost`, `errsendpost`

A channel program might be unable to deliver a message because of long-term service failures or invalid addresses. When this occurs, the MTA 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 an excessive amount of traffic with which the postmaster must deal. (See Keywords for Sending Notification Messages to the Postmaster and Sender.)

Warning Messages

Keywords: `warnpost`, `nowarnpost`, `copywarnpost`, `errwarnpost`

In addition to returning messages, the MTA can send detailed warnings for undelivered messages. This is generally due to time-outs 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 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 can be 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. (See Keywords for Sending Notification Messages to the Postmaster and Sender.)

Blank Envelope Return Addresses

Keywords: `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 the MTA 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 envelope From: addresses and may require the use of this option.

Bit 1 (value = 2) controls whether or not the MTA replaces all blank envelope addresses with the address of the local postmaster. This is used to accommodate noncompliant systems that do not conform to RFC 821, RFC 822, or RFC 1123.

Bit 2 (value = 4) prohibits syntactically invalid return addresses.

Bit 3 (value = 8) same as `mailfromdnsverify` keyword.

Postmaster Returned Message Content

Keywords: `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. (See Keywords for Sending Notification Messages to the Postmaster and Sender.)

Setting Per Channel Postmaster Addresses

Keywords: `aliaspostmaster`, `returnaddress`, `noreturnaddress`, `returnpersonal`, `noreturnpersonal`

By default, the Postmaster's return address that is used when the MTA constructs bounce or status notification messages is `postmaster@`*local-host*, where *local-host* is the official local host name (the name on the local channel), and the Postmaster personal name is "MTA e-Mail Interconnect." Care should be taken in the selection of the Postmaster address. An illegal selection can cause rapid message looping and a great number of error messages.

The `RETURN_ADDRESS` and `RETURN_PERSONAL` options can be used to set an MTA system default for the Postmaster address and personal name. Or, if per channel controls are desired, the `returnaddress` and `returnpersonal` channel keywords may be used. `returnaddress` and `returnpersonal` each take a required argument specifying the Postmaster address and Postmaster personal name, respectively. `noreturnaddress` and `noreturnpersonal` are the defaults and signify that the default values should be used. The defaults are established via the `RETURN_ADDRESS` and `RETURN_PERSONAL` options or the normal default values if such options are not set.

If the `aliaspostmaster` keyword is placed on a channel, then any messages addressed to the user name `postmaster` (lowercase, uppercase, or mixed case) at the official channel name is redirected to `postmaster@`*local-host*, where *local-host* is the official local host name (the name on the local channel). Note that Internet standards require that any domain in the DNS that accepts mail have a valid postmaster account that receives mail. So this keyword can be useful when it is desired to centralize postmaster responsibilities, rather than setting separate postmaster accounts for separate domains. That is, whereas `returnaddress` controls what return postmaster address is used when the MTA generates a notification message from the postmaster, `aliaspostmaster` affects what the MTA does with messages addressed to the postmaster.

Keywords for Sending Notification Messages to the Postmaster and Sender

Keyword	Description

Returned Message Content	*Specifies Addresses on Notifications*
`notices`	Specifies the time that may elapse before notices are sent and messages returned.
`nonurgentnotices`	Specifies the time that may elapse before notices are sent and messages returned for messages of non-urgent priority.
`normalnotices`	Specifies the time that may elapse before notices are sent and messages returned for messages of normal priority.
`urgentnotices`	Specify the time which may elapse before notices are sent and messages returned for messages of urgent priority.
Returned Messages	*How to handle failure notices for returned messages.*
`sendpost`	Enables sending a copy of all failed messages to the postmaster.
`copysendpost`	Sends a copy of the failure notice to the postmaster unless the originator address on the failing message is blank, in which case, the postmaster gets copies of all failed messages except those messages that are actually themselves bounces or notifications.
`errsendpost`	Sends a copy of the failure notice to the postmaster only when the notice cannot be returned to the originator. If `nosendpost` is specified, failed messages are never sent to the postmaster.
`nosendpost`	Disables sending a copy of all failed messages to the postmaster.
Warning Messages	*How to handle warning messages.*
`warnpost`	Enables sending a copy of warning messages to the postmaster. The default 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.
`copywarnpost`	Sends a copy of the warning message to the postmaster unless the originator address on the undelivered message is blank.
`errwarnpost`	Sends a copy of the warning message to the postmaster when the notice cannot be returned to the originator.
`nowarnpost`	Disables sending a copy of warning messages to the postmaster.
Returned Message Content	*Specifies whether to send entire message or just headers to the postmaster.*
`postheadonly`	Returns only headers to the postmaster. Restricting the postmaster copy to just the headers adds an additional level of privacy to user mail. However, this does not guarantee message security as postmasters and system managers are able to read the contents of messages using `root` system privileges, if they choose.
`postheadbody`	Returns both the headers and the contents of the message.
Returned Message Content	*Specifies Addresses on Notifications*
`includefinal`	Include final form of address in delivery notifications (recipient address).

`returnenvelope`	Control use of blank envelope return addresses. 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 the MTA 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.Bit 1 (value = 2) controls whether or not the MTA replaces all blank envelope addresses with the address of the local postmaster. This is used to accommodate noncompliant systems that do not conform to RFC 821, RFC 822, or RFC 1123.Bit 2 (value = 4) prohibits syntactically invalid return addresses.Bit 3 (value = 8) same as `mailfromdnsverify` keyword.
`suppressfinal`	Suppress the final address form from notification messages, if an original address form is present, from notification messages.
`useintermediate`	Uses an intermediate form of the address produced after list expansion, but prior to user mailbox name generation. If this information isn't available, the final form is used.
Returned Message Content	*Specifies Addresses on Notifications*
`aliaspostmaster`	Messages addressed to the user name postmaster at the official channel name is redirected to postmaster@local-host, where local-host is the local host name (the name on the local channel).
`returnaddress`	Specifies the return address for the local postmaster.
`noreturnaddress`	Use `RETURN_ADDRESS` option value as postmaster address name.
`returnpersonal`	Set the personal name for the local postmaster.
`noreturnpersonal`	Use `RETURN_PERSONAL` option value as postmaster personal name.

Controlling Message Disposition Notifications

Message Disposition Notifications (MDN) are email reports sent by the MTA to a sender and/or postmaster reporting on a message's delivery disposition. For example, if a message is rejected by a Sieve filter, an MDN will be sent to the sender. MDNs are also known as read receipts, acknowledgements, receipt notifications, or delivery receipts. The Sieve scripting language is typically used for messaging filtering and vacation messages.

To Customize and Localize Message Disposition Notification Messages

The instructions for modifying and localizing MDNs parallel those described in customizing and localizing delivery status notification messages with some minor differences as described here. (See To Customize and Localize Delivery Status Notification Messages and Internationalization of Generated Notices.)

The mapping (called the `DISPOSITION_LANGUAGE` mapping) parallels the `notification_language` mapping table (see To Customize and Localize Delivery Status Notification Messages) used to internationalize status notifications.

However, probes for MDNs to this mapping take the following form:

```
type|modifiers|source-channel|header-language|return|recipient
```

Where:

`type` is disposition type, which can be one of the following: `displayed`, `dispatched`, `processed`,

deleted, denied, or failed.

modifiers is a comma-separate list of disposition modifiers. The current list is: error, warning, superseded, and expired.

source-channel is the source channel producing the MDN.

header-language is the language specified in one of the following: accept-language, preferred-language, or x-accept-language. (MTA uses the first of these options that is present.)

return is the address to which the notification is being returned.

recipient is the address that the disposition is about.

The result of the disposition mapping consists of two or three pieces of information separated by vertical bars (|). The first piece of information is the directory where the template files for the disposition notification can be found. The second piece of information is the character set into which the standalone disposition text should be forced. (This information is required because some dispositions. notably the dispositions produced by autoreply echo or the use of the :mime parameter to the vacation Sieve action, do not employ template files and consequently cannot inherit the character set from those files.) Finally, the third piece of information is an override subject line for the notification. This information is only used if the $T flag is also set by the mapping.

The following additional template files are used to construct MDNs:

```
disposition_deleted.txt disposition_failed.txtdisposition_denied.txt
disposition_prefix.txtdisposition_dispatched.txt
disposition_processed.txtdisposition_displayed.txt
disposition_suffix.txtdisposition_option.opt
```

The use of these template files parallels that of the various return_*.txt files for status notification messages. Message text for *.txt files should be limited to 78 characters per line.

Optimizing MTA Performance

This section describes miscellaneous optimizations for the MTA. It consists of the following section:

- Optimizing Authorization Checks to the LDAP Directory for Messages Addressed to Mailing Lists

Optimizing Authorization Checks to the LDAP Directory for Messages Addressed to Mailing Lists

You can use metacharacter character substitutions to reduce authorization checks to the LDAP directory for Messages addressed to a mailing list.

Metacharacter substitutions can now be specified in mgrpModerator, mgrpAllowedBroadcaster and mgrpDisallowedBroadcaster attributes. In particular, the various address-related metacharacter sequences ($A for the entire address, $U for the mailbox part, $D for the domain part) refer to the current envelope From: address and can in some cases be used to limit the results returned by the URL to entries that are likely (or guaranteed) to match. This may make authorization checks much more efficient.

The new MTA option PROCESS_SUBSTITUTIONS controls whether or not substitutions are performed in various LDAP attributes that specify a URL. This is a bit-encoded value, with the bits defined as shown in the following table:

Bit	Value	Description
0	1	Enables substitutions in `mgrpDisallowedBroadcaster` if set
1	2	Enables substitutions in `mgrpAllowedBroadcaster` if set
2	4	Enables substitutions in `mgrpModerator` if set
3	8	Enables substitutions in `mgrpDeliverTo` if set
4	16	Enables substitutions in `memberURL`

The `PROCESS_SUBSTITUTIONS` MTA option defaults to 0, meaning that all of these substitutions are disabled by default.

An example would be a dynamic list defined through an LDAP lookup where anyone on the list is allowed to post. In such cases you typically define the list with attributes like:

```
mgrpAllowedBroadcaster:
ldap:///o=Sesta,c=US??sub?(&amp;(objectClass=inetMailUser)(objectClass=inetOrgP
```

The effect of such a definition, however, is to expand the list twice, once for the authorization check and once to build the actual recipient list,. This is a very server intensive operation. If, on the other hand, you add a restriction so only entries containing the current envelope `From:` address are returned in the authorization check, things may be much more efficient. First change the `PROCESS_SUBSTITUTION` setting to 2, and then you can set the following entries:

```
mgrpAllowedBroadcaster:
ldap:///o=Sesta,c=US??sub?(&amp;(objectClass=inetMailUser)(objectClass=inetOrgP
```

In this example, only the sender's entry is checked for broadcast authorization as opposed to all the user entries in Sesta US. This reduces the work the directory server has to do to a single (hopefully indexed) match and a single return value. The alternative is to return the entire list and have the MTA perform the match.

Note that the information available for substitution varies depending on whether the attribute is used for authorization checks or for actual list expansion. For authorization attributes, the whole address (`$A`), domain (`$D`), host (`$H`), and local-part (`$L`) are all derived from the authenticated sender address. In the case of list expansion attributes all of these substitution values are derived from the envelope recipient address that specified the list. In both cases, however, the subaddress substitution (`$S`) is derived from the current envelope recipient address.

The ability to access subaddress information in list expansion URLs makes it possible to define *metagroups*, that is, a single group entry that creates an entire collection of different groups. For example, a group with a `mgrpDeliverTo` value of:

```
mgrpDeliverTo: ldap:///o=usergroup?mail?sub?(department=$S)
```

and the corresponding `PROCESS_SUBSTITUTIONS` value being 8. It is possible to send mail to every member of a given department with an address of the form `group+department@domain.com`. Note that a mechanism like a forward mapping could be used to alter the syntax if subaddresses are seen as too difficult.

Chapter 9. Messaging Server Features

Messaging Server Features

These are Messaging Server features that have not yet been incorporated into other documentation.

Features Introduced in Messaging Server 6.3 P1

Features Introduced in Messaging Server 6.3 P1

XML logging format defaults changed

The new XML format for mail.log is much more tolerant of extensions and is a new format in any case. In the past our choice of defaults for MTA options controlling the addition of information to mail.log has been based on the fact that any change might break existing parsers. This is not a concern for XML, where the rule needs to be "ignore any attribute you don't understand". It therefore makes sense to change the defaults for various optional bits of log information when XML format is used. The following options

```
LOG_FILENAME
LOG_FILTER
LOG_MESSAGE_ID
LOG_NOTARY
LOG_PRIORITY
LOG_PROCESS
LOG_QUEUE_TIME
LOG_REASON
LOG_USERNAME
```

will now default to 1 if XML format is enabled. The old default of 0 will be used with any other format.

Additional information on logging format settings available in the following pages:

Specifying Additional MTA Logging Options
SMTP (TCP/IP) Channel Option Files

Support for Sender Rewriting Scheme (SPF) added.

Handling Forwarded Mail in SPF Using the Sender Rewriting Scheme (SRS)

Mapping table template maximum size increase

The maximum size of a mapping template has been increased from 256 to 1024 characters. The maximum size of a line in the mapping file has also been increased from 1024 to 4096 characters.

Sieve redirect enhancements

- Sieve redirect can now add three header fields:

```
resent-date: <date-of-resend-operation>
 resent-to:   <address-specified-in-redirect>
 resent-from: <addres-of-sieve-owner>
```

  The new :resent and :noresent arguments to redirect can be used to control whether or not these fields are added. If neither argument is specific the system default is used. The system default is controlled by the new SIEVE_REDIRECT_ADD_RESENT MTA option. Setting the option to 1 causes these fields to be generated unless :noresent used. A setting of 0 causes the fields to be generated only if :resent is used. The option defaults to 1, which means the fields are generated by default for regular redirects.

Related Change Request
RFE# 6545463 - "Add the ability for sieve redirect to generate appropriate resent- fields automatically"

- Sieve redirect has been enhanced with three new arguments:

```
:resetmailfrom - Reset the envelope from (MAIL FROM) address to
that of the current Sieve owner.
 :keepmailfrom  - Preserve the envelope from (MAIL FROM) address
from the original message.
 :notify        - Specify a new set of notification flags for the
redirected message. A single
                  parameter is required giving a list of
notification flags. The same set of
                  flags accepted by the NOTIFY parameter of the DSN
SMTP extension are accepted
                  here: SUCCESS, FAILURE, DELAY and NEVER. Note
that these falgs are specified
                  as a Sieve list, e.g.
                  redirect :notify ["SUCCESS","FAILURE"]
"foo@example.com";
```

  The default if :notify isn't specified is the normal SMTP default of FAILURE,DELAY. :keepmailfrom is the default unless :notify is specified, in which case the default switches to :resetmailfrom. The one additional exception is that specification of the SUCCESS flag forces the use of :resetmailfrom unconditionally.

Related Change Request
RFE# 6500463 - "Add :envelope-from argument to Sieve redirect action; add :notification-flags argument"

New USERNAME_MAPPING spam filter option

A new spam filter option USERNAME_MAPPING has been added to the SpamAssassin plugin. Further details are available here:

SpamAssassin Options

Related Change Request
RFE# 6489137 - "Allow the username sent to SpamAssassin's spamd to be set based on recipient address information"

New "imsimta qm jobs" command

Add the command "imsimta qm jobs". Here's the doc from the online help files:

```
Syntax: jobs [-[no]hosts] [-[no]jobs] [-[no]messages] [channel-name]
```

Channel by channel, list all active and pending delivery processing jobs currently being managed by the Job Controller. Additional cumulative information is provided for each channel such as the number of message files successfully delivered and those requeued for subsequent delivery attempts.

When the optional channel-name parameter is omitted, all channels are listed. The channel-name parameter may contain the * and ? wild card characters.

The `-nohosts` and `-nomessages` switches may be used, respectively, to suppress the line by line output of individual destination hosts and messages. The default is `-hosts` and `-messages`.

When `-nojobs` is specified then only the basic information is listed for each channel. The individual processing jobs, destination hosts, and messages are not listed. That is `-nojobs` implies `-nohosts` and `-nomessages`.

"imsimta qm messages" command removed

"imsimta summarize" command updated to provide "imsimta qm messages" functionality as well as additional output options. Formatting changes were made to the output of all variants of "imsimta summarize".

As a result the redundant "imsimta qm messages" command has been removed.

What follows is the online help for "imsimta summarize".

```
summarize

 Syntax: summarize [-database|directory_tree] [-[no]heading] [-[no]held]
                   [-[no]hosts] [-[no]oldest] [-[no]trailing]
[channel-name]

 Display a summary listing of queued messages by channel.  By default,
the
 Messaging Server's on-disk directory tree is used as the source of
queued
 message information; this default may be changed with the view command.
The
 -database and -directory_tree switches may be used to override the
default.
 The -database switch selects the in-memory queue database held by the
Job
 Controller.

 The channel-name parameter may be used to restrict the listing to one
or more
 channels.  If the channel-name parameter is omitted, a listing is made
for
 all channels.  The channel name parameter may contain * and ? wild card
 characters.

 The -oldest switch may be used display per channel the oldest queued
message
 files.  Combining this switch with -database will cause the Job
Controller to
 be queried about each and every message file for each channel
summarized.

 The -hosts switch, which implies -database, will additionally display
per
 destination host queueing information held by the Job Controller for
the
 SMTP protocol based channels.

 By default, active messages are listed.  Specify -held to instead list
 messages which have been marked as held.  Note that -held implies
 -directory_tree.
```

New LOG_PAGE_COUNT option added to one-way SMS channel

The following newoption has been added to the one-way SMS channel:

```
LOG_PAGE_COUNT (0, 1, 2)
The LOG_PAGE_CONT channel option only takes effect when logging is
enabled
for the channel with the "logging" channel keyword.  When logging is
enabled,
this option controls the value recorded in the mail.log file's message
size
field.  Normally that field gives the block size of the underlying
message
file.  When LOG_PAGE_COUNT has a non-zero value, the number of
transmitted
pages will instead be recorded in that field of the log file.

0 -- Log the block size of the underlying message file.  This is the
     default behavior when LOG_PAGE_COUNT is not specified.

1 -- Log the count of pages sent when the entire message is successfully
     transmitted to the recipient.  Otherwise, log a page count of zero
     even if some pages were sent to the recipient.

2 -- Log the count of pages sent to the recipient regardless of whether
     or not the entire message was sent.

The distinction between LOG_PAGE_COUNT=1 and LOG_PAGE_COUNT=2 is only
relevant
when a message is sufficiently large that it will be transmitted as
several
pages.  In that case, it is possible that before transmitting all the
pages
an error might occur.  For example, the network between the MTA and the
remote SMPP server goes down.  In that case, a later retransmission
attempt
will be made for the message. For each attempt, the previously sent
pages
are sent again along with the pages which were not sent.  Sites may
choose
whether or not they want to record the count of pages successfully sent
during
these failed delivery attempts.
```

Related Change Request
RFE# 5107406 - "add logging capabilities for multi-page sms's"

Comment introducer character '#' available for imsimta qm

imsimta qm command lines now accept the character '#' as a comment introducer. Command lines may now either be a valid command or a comment line. The two may not be mixed on the same line. This functionality is useful for documenting command scripts used with qm via qm's "run" command.

Related Change Request
RFE# 6546112 - "Provide an option to add comments in file passed to imsimta qm run"

Additional `After_Auth` state for mailing list deferred processing

The mailDeferProcessing attribute will now accept a value besides "Yes" and "No". This new value, "After_Auth", causes group validation to be done immediately but group expansion to be deferred.

The related DEFER_GROUP_PROCESSING MTA option now accepts the value -1, meaning "After_Auth" will be the default for groups without a mailDeferProcessing attribute.

Note that one side effect of this new setting is that authorization checks will be performed twice, once on initial submission and once by the reprocess channel. It is not feasible to bypass the reprocess channel recheck because there are tricky ways to route around the initial check.

> **Related Change Request**
> RFE# 6554522 - "Need a way to perform group access checks but then defer actual group expansion"

New $~ metacharacter for FROM_ACCESS mapping

The $~ metacharacter, when specified in a FROM_ACCESS mapping, will cause a channel name to be read from the template string (immediately following any debug level setting done with $U). Use of $~ causes the enqueue initialization process to essentially restart because a lot of it can depend on the source channel. This includes re-evaluation of the FROM_ACCESS mapping with the new source channel in place. A flag is used to prevent this process from happening more than once - $~ is simply ignored if it is encountered on a second pass through the FROM_ACCESS mapping.

> **Related Change Request**
> RFE# 6554529 - "FROM_ACCESS mapping should be able to perform a source channel switch"

Features Introduced in Messaging Server 7.0

Features Introduced in Messaging Server 7.0

New setting for PROCESS_SUBSTITUTIONS MTA option.

An additional bit has been defined in PROCESS_SUBSTITUTIONS. Bit 5 (value 32), if set, enables substitution processing on `mgrpErrorsTo` attribute values. In this case the information available for substitution comes from the address of group itself, not the current return address or any group expansion addresses.

New mapping table pattern metacharacter ($C) to match ASCII control characters

$C has been added to the list of mapping file pattern match metacharacters. $C matches any ASCII control character other than horizontal tab.

New MAPPING_PARANOIA MTA option

Since *_ACCESS mapping tables use vertical bars as delimiters issues can arise when externally provided material like envelope from or to addresses themselves contain vertical bars. This option is intended to provide various tools to handle such issues. Giving this option a non-negative value will cause any vertical bars in various mapping input strings to be replaced with the character whose ASCII value is given by this option in the mapping probe. A negative value will cause the vertical bar to simply be dropped entirely from the probe. The default value for this option is 124, the ASCII value for vertical bar, which causes vertical bars to be left untouched.

Additionally, the $| metacharacter flag will be set unconditionally when any vertical bars are detected in external material used to construct *_ACCESS mapping probes. $:| can then be used to test to see if one or more of the vertical bars present in the probe string are not probe delimiters.

New AUTH_REWRITE mapping table option ($A) to add a message header

$A in an AUTH_REWRITE mapping now works the same way as $A in the various *_ACCESS mappings: It takes a single argument and interprets it as a header to add to the primary message header. The argument position is after $Z processing (forced From: field) but before $N processing (fail message).

New PORT_ACCESS mapping table option ($S) to switch source channel

The PORT_ACCESS mapping done in the SMTP server now has the ability to change the current source channel. Specifically, setting $S will cause an additional argument to be read from the mapping result and interpreted as a channel name. This argument must appear after any protocol delay argument.

New USE_AUTH_RETURN MTA option

A new MTA option, USE_AUTH_RETURN, has been added. This option is bit-encoded with the various bits matching those of the USE_ORIG_RETURN option. Each place where the MTA performs a comparison operation against the envelope from (MAIL FROM) address has an assigned bit. If the bit in USE_AUTH_RETURN is clear normal rewriting is applied to the address selected by the USE_ORIG_RETURN and USE_CANONICAL_RETURN MTA options. If, however, the bit is set and an authentication has produced an authenticated sender address, the authenticated sender address will be used instead.

It should be noted that if USE_AUTH_RETURN causes the use of an authenticated address it will make the corresponding bit in USE_CANONICAL_RETURN and USE_ORIG_RETURN noops.

New metacharacter flag ($S) for sieve FILTER_* mapping tables

The $S metacharacter flag will be set in sieve mapping callouts (FILTER_*) if the callout is being done from a system-level sieve.

Unknown channel keyword behaviour modified

All support for rightlist identifier access control has been removed from the MTA. Unknown channel keywords will now result in a configuration error rather than being treated as a rightslist identifier.

Support for future release SMTP extension and enhancements to Deferred-delivery header mechanism

Support for the future release SMTP extension defined in RFC 4865 has been added. The future release SMTP extension is part of Lemonade Profile 1 Support. This support is enabled by placing the `futurerelease` channel keyword on the source channel used for initial message submission.

The keyword takes a single integer argument: The maximum number of seconds a message can be held.

Care should be used when enabling future release since it allows messages to be in effect stored in the MTA's queues. Future release should only be used for channels handling initial message submission and authentication should be required.

Note that similar functionality was previously available: Specification of a Deferred-delivery header field in a submitted message coupled with use of the deferred channel keyword on the destination channel provided the ability to defer delivery of messages. However, future release provides superior functionality for the following reasons:

1. The facility is controlled by a setting on the source channel, allowing it to be provided to a subset of the user population. Placing deferred on a destination channel opened the door to anyone submitting a message to that channel that would be held for some period of time.
2. There's no way for a client which sets a Deferred-delivery header field to know whether or not the header has actually caused the message to be deferred. The future release SMTP extension, on the other hand, lets the client know how long a message can be held and an error will be returned to the client if the message cannot be held for the time the client wants.

3. There was no way to place a limit on the amount of time a message could be held. Instead what happened was a message held longer than the channel's last notices value would simply be returned as undeliverable.
4. Deferred-delivery settings on messages did not survive a job controller restart.

As part of the implementation work for future release the old Deferred-delivery: mechanism has been redesigned to address some (but not all) of these points. In particular, the deferred channel keyword has been replaced by two new channel keywords, `deferredsource` and `deferreddestination`. (The deferred keyword is now a synonym for deferreddestination.) Both of these keywords accept an optional integer argument specifying in seconds the maximum amount of time in the future a Deferred-delivery: header can specify and still be honored. The default if no argument is specified is 60*60*24*7, or 7 days. `Deferredsource` enables Deferred-delivery: processing on the basis of the source channel while `deferreddestination` operates on destination channels. Finally, Deferred-delivery settings on messages now survive job controller restarts. This addresses all of the points on the above list except (2) - use of a Deferred-delivery: header field still provides no mechanism for informing the client whether or not the setting will be honored.

Related Change Request
RFE #4669529 - "SMTP Future Delivery"

Obsolete `kanji_dec` and `kanji_jis` channel keywords removed

The obsolete kanji_dec and kanji_jis channel keywords have been removed. Appropriate charset7 settings should be used instead.

Enhancement to MTA duplicate recipient elimination

Duplicate recipient elimination checking has been enhanced to consider the content of any added headers. Previously if two different header additions were performed with the same material for the same recipient the extra copy would not necessarily be eliminated.

Related Change Request
RFE #6570005 - "Multiple copies of message on redirect"

mailUserStatus and inetUserStatus status setting priorities can now be reordered

Mail status (`mailUserStatus`) and `inetMailGroupStatus` setting values will now be honored if they are used as general status settings (inetuserstatus and inetgroupstatus). The intent of this change is not to encourage direct use of such status settings, but rather so that the priorities of the two status settings for users can be reordered by setting the following MTA options:

```
LDAP_USER_STATUS=mailUserStatus
LDAP_USER_MAIL_STATUS=inetUserStatus
```

MTA limit on maximum UID length

The MTA now imposes a limit of 128 octets on UIDs. A UID longer than this limit will result in the user entry being considered invalid. This is being done because various lower layer libraries have hard buffer limits that preclude longer UIDs.

MTA stressed channel concept added to job controller

Added the concept of a stressed channel to the job controller to provide Message Store Load Throttling.

Channel programs can now tell the job controller if they are being overwhelmed. If they do that then the job controller checks to see if we have been told about this recently: the job controller ignores stressed channel messages that are received within `StressBlackout` seconds of a previous stressed message for the same channel. If the message is processed, then the job controller will multiply the effective `threaddepth` parameter for the channel by `StressFactor`, and subtract `StressJob` from the job limit for the channel. `Threaddepth` never goes over 134,217,727, and job limit never goes below 1. In addition, then job controller will ask all current master programs for the channel to exit, and will, if the queue is not empty, start an appropriate number of processes.

`stressTime` seconds after the last stress change, the job controller divides `threaddepth` by `UnstressFactor` (never allowing thread depth to drop below the original configured `threaddepth`), and adds `UnstressJob` to the job limit (never allowing the job limit to rise above the original configured limit. A "stress change" is either an increase in stress or a decrease in stress.

New sub-commands have been added to the "imsimta qm" utility:

```
stress <channel name>
unstress <channel name>
```

The qm stress command is treated the same way as another advice to the job controller about a channel being stressed: repeated use of this command within `StressBlackout` for the same channel will be ignored. This blackout does not apply to the `unstress command` which can be issued repeatedly if necessary to return a very stressed channel to its original state.

These configuration parameters go in the global section of the job controller configuration file. The default values are:

```
StressBlackout=60
StressTime=120
StressFactor=5
StressJobs=2
UnstressFactor=StressFactor
UnstressJobs=StressJobs
```

Support for BURL submit extension

Support has been added to the MTA for the BURL submit extension (RFC 4468) as part of Lemonade Profile 1 Support.

New USE_IP_ACCESS MTA option

New bit-encoded MTA option USE_IP_ACCESS. Currently only bit 0 (value 1) is defined - if set an additional field is added to the end of the IP_ACCESS probe string: The retry count for the current message, as determined by the message file name. A -1 will appear if the retry count cannot be determined. This information can be used to change the retry strategy based on the number of delivery attempts.

Sieve support for ereject action added and behaviour of reject, refuse actions modified

Support for the Internet Draft draft-ietf-sieve-refuse-reject-06.txt has been implemented. The behavior of reject has changed to match what the draft specifies and support for ereject has been added. Additionally, the refuse action no longer unconditionally produces an SMTP-level reject - this will only happen if the refuse applies to all recipients. Refuse now had semantics similar to jettison, in that a refuse done in a more general (e.g., system-level) sieve will override actions taken by more specific (e.g., user-level) sieves.

Related Change Request
RFE #6667527 - "Implement support for Sieve ereject, update semantics of existing reject and refuse actions"

Wildcard support added to 552_PERMANENT_ERROR_STRING MTA option

The 552_PERMANENT_ERROR_STRING option is used to make exceptions in the handling of 552 responses to RCPT TO. (Due to a bug in RFC 821 that miscategorized 552 as a temporary error, RFC 2821 recommends treating 552 responses as temporary errors so that is how Messaging Server behaves.) However, some of these responses include the input address, which cannot be matched with a simple string comparison. Glob-style wildcards are needed and have now been implemented.

Related Change Request
RFE #6667935 - "Allow wildcards in 552_PERMANENT_ERROR_STRING TCP channel option"

SMTP status line now includes envelope ID

Some message tracking systems look at the final SMTP status line for an identifier to use for "next hop" tracking. Although iMS doesn't have queue identifiers that can be used for this purpose, there's no reason why the envelope identifier (NOTARY ENVID) cannot serve a similar, if not superior, function. Accordingly, the final SMTP status now includes the envelope id.

For example this is the response provided by Messaging Server 6.3:

```
250 2.5.0 Ok.
```

This is the response now provided by Messaging Server 7.0:

```
250 2.5.0 Ok, envelope id 0KIU0090CLPKVO00@sun.com
```

Related Change Request
RFE #6667951 - "Return effective envelope identifier in the SMTP end of message status response"

Domain level sieves now applied when domain level routing is used

Setting mailRoutingHost for a domain when the ROUTE_TO_ROUTING_HOSTS MTA option is set to 1 causes user lookups for the domain to be bypassed and forced routing is applied to direct the message to the specified routing host. In this situation domain optins to spam filters are still applied. Domain sieves, however, were not. This limitation has now been addressed - domain sieves will be attached to recipient addresses when domain level routing is employed.

Setting $V In PORT_ACCESS Mapping Enables Private SMTP Extensions

Setting $V in a `PORT_ACCESS` mapping enables the MTA's private SMTP extensions `XADR`, `XCIR`, `XGEN`, and `XSTA`, overriding any SMTP server `DISABLE_*` channel options.

Features Introduced in Messaging Server 7 Update 1

Features Introduced in Messaging Server 7 Update 1

New flag ($V) for MAIL_ACCESS, ORIG_MAIL_ACCESS, SEND_ACCESS, and ORIG_SEND_ACCESS mapping tables

The $V flag will be set in the MAIL_ACCESS, ORIG_MAIL_ACCESS, SEND_ACCESS, and ORIG_SEND_ACCESS mappings if the current address was arrived at via alias expansion.

Additional functionality added to REVERSE mapping table

The $C flag will be set in the REVERSE mapping if the reversal operation is attempting to produce a canonical address for MTA use in comparison operations. The $M flag will be set if the reversal operation is attempting to produce a reversed version of the `MAIL FROM` address for use in address validity checks when the canonical form has not been selected.

The REVERSE mapping result is now interpreted as a series of addresses separated by commas. The first address becomes the reversal result if the $Y flag is set by the mapping. If both $Y and $H are set and input is the `MAIL FROM` address the second address becomes the default postmaster address for this sender.

New SMTP channel timeout options

New SMTP channel options HELLO_RECEIVE_TIME, SESSION_TIME, and TRANSACTION_TIME. Previously COMMAND_RECEIVE_TIME specified the amount of time the server would wait for a command.

HELLO_RECEIVE_TIME now specifies the amount of time that the server will wait for the initial HELO/EHLO from the client. The timeout for all other commands is still controlled by COMMAND_RECEIVE_TIME.

TRANSACTION_TIME specifies the maximum amount of time, in minutes, a SMTP transaction is allowed before the session will be disconnected. It defaults to 600 (10 hours). Note that this value is checked only once every 10 reads (in order to save on time() calls).

SESSION_TIME specifies the maximum amount of time, in minutes, a SMTP session is allowed before being disconnected. It defaults to 1200 (20 hours). This one is also checked only once every 10 reads.

New `disconnectcommandllimit` channel keyword added

New channel option `disconnectcommandllimit`. SMTP sessions using more than this number of commands will be disconnected. The default is infinite.

Limited SASL capabilities added to MTA SMTP client

SASL authentication with the destination MTA will be attempted if the `maysaslclient` or `mustsaslclient` channel options are set (it must succeed if `mustsaslclient` is set).

The PLAIN and EXTERNAL SASL mechanisms are currently supported. The `AUTH_USERNAME` and `AUTH_PASSWORD` TCP channel options provide the credentials for the plain mechanism and the `EXTERNAL_IDENTITY` TCP channel option provides the identity string for `SASL EXTERNAL`. (`EXTERNAL_IDENTITY` can be set to the empty string to enable SASL EXTERNAL without an identity string.)

The following steps provide an example of using the `mustsaslclient` keyword to enforce SASL authentication to the destination MTA.

1. Create a new channel which uses the `mustsaslclient` keyword in the Messaging Server `imta.cnf` configuration file e.g.

```
! tcp_secure
tcp_secure nomx multiple subdirs 20 dequeue_removeroute maxjobs 7
musttlsclient mustsaslclient \
pool SMTP_POOL daemon [192.168.1.10]
tcp_secure-daemon
```

> **Note**
> The `musttlsclient` keyword ensures that the connection is encrypted prior to passing across the plain-text username/password. The example tcp_secure channel is also configured to send all emails to the MTA with an IP address of 192.168.1.10.

2. Create a rewrite rule in the Messaging Server `imta.cnf` configuration file which directs emails for some.address@domain.com to the new channel.

```
domain.com $U%$D@tcp_secure-daemon
```

3. Create a `tcp_secure_option` TCP channel option configuration file in the Messaging Server configuration directory which contains the SASL authentication username and password which is known to the MTA system at 192.168.1.10.

```
bash-3.00# cat /opt/sun/comms/messaging/config/tcp_secure_option
AUTH_USERNAME=authuser
AUTH_PASSWORD=secretpassword
```

4. Rebuild the Messaging Server MTA configuration database and restart the MTA.

```
./imsimta cnbuild
./imsimta restart
```

5. Send an email to some.address@domain.com. You should see a log entry in `mail.log_current` resembling the following entry:

```
27-Jan-2009 12:25:51.82 tcp_secure                DEQS 1
my.address@internal.com rfc822;some.address@domain.com
some.address@domain.com [192.168.1.10] dns;[192.168.1.10]
(TCP|1.2.3.4|34682|192.168.1.10|25) (domain.com --
Server ESMTP [Sun Java System Messaging Server 6.2-9.14 [built Aug
19 2008]]) smtp;250 2.1.5 address accepted for
deferred processing: some.address@domain.com
```

Note
DEQS stands for "(D)equeue, (E)SMTP, TL(S)/SSL used". (Q) indicates that SMTP PIPELINING was used (RFC 1854)

New MTA options SPAMFILTERX_RETURNPATH (X=1-8)

New MTA options SPAMFILTERX_RETURNPATH (X=1-8). If set to 1 this will cause a `Return-path:` header to be added to the message passed to the corresponding filter. 0, the default, disables this addition.

Additional verification functionality added to `imsimta test -domain` command

The domain entry verification facility has been enhanced to perform some additional checks. First, if a default `mailhost` attribute is defined at the domain level it will now be checked. Second, so-called short-form domain names (names without a period separator) now generate warnings. Third, some additional cross checks on domains versus aliases are now performed. The following additional messages can now be output:

```
%DMAP-E-ALIASANDBASEDN, Domain alias entry '%.*s' also contains a base
DN pointer
  %DMAP-W-EMPDEFMAILHOST, Domain '%.*s' has an empty mailHost default
  %DMAP-E-INVALIDDEFMAILHOST, Default mailHost '%s' for domain '%.*s' is
invalid
  %DMAP-W-SHORTFORMDEFMAILHOST, Default mailHost '%s' for domain '%.*s'
is a shortform name
  %DMAP-E-MULTIDEFMAILHOST, Multivalued mailHost default in entry for
domain '%.*s', used value '%s' ignored '%s'
  %DMAP-E-DOMAINALIASINVALID, Domain alias '%.*s' defined/referenced by
domain entry with DN '%.*s'
    is syntactically invalid
  %DMAP-W-SHORTFORMDOMAIN, Short form domain name '%.*s'
defined/referenced by domain entry with DN '%.*s'
  %DMAP-W-SHORTFORMDOMAINALIAS, Short form domain alias '%.*s'
defined/referenced by domain entry with DN '%.*s'
```

Sieve environment extension support

The Sieve environment extension specified in RFC 5183 has been implemented.

All of the items defined in the RFC are provided. Additionally, the `vnd.sun.source-channel` item returns the name of the current source channel and the `vnd.sun.destination-channel` item returns the name of the current destination channel.

Sieve extlists extension support

Support has been added for the Sieve external lists (extlists) extension. This extension is described in the Internet-Draft:

http://tools.ietf.org/id/draft-melnikov-sieve-external-lists-01.txt

It is important to note, however, that the implementation differs from what's specified in the draft RFC in several ways:

1. `:list` is considered to be a new match type. This means it cannot be combined with existing match types like `:is`, `:contains`, and `:matches`. The draft considers `:list` to be an independent argument, but since it doesn't make sense to combine `:list` with any other match type, this grouping seems sensible.
2. The draft says that `:list` can be used with three existing tests: *envelope*, *header*, and *address*. In the Sun Messaging Server implementation `:list` is more generally available and can be used with these tests:

```
address
currentdate
date
deleteheader
envelope
environment
hasflag
header
replaceheader (nonstandard)
spamtest
string
virustest
```

   Note that *replaceheader* and *deleteheader* are actions, not tests. However, they both include a test subcomponent that borrows existing header test machinery. It is this subcomponent that now accepts `:list`.

3. The draft RFC does not provide support for properties associated with list entries - a very useful feature. We have implemented support for this through the Sieve variables extension. If variables are enabled `:list` sets variables in a fashion similar to `:matches` : ${0} is set to whatever was found on the list, ${1} is the first property value associated with the list entry, ${2} is the second, and so on.
4. While the redirect `:list` "URL" form can be supported by our implementation, doing so would be extremely dangerous and is **NOT recommended**.

The external lists extension is implemented through a new mapping, `SIEVE_EXTLISTS`. The mapping's existence enables the `extlists` extension. When `:list` is specified as part of one of the tests listed above it causes the mapping to be called with a probe string of the form:

```
test-name|owner|spare_4|spare_5|spare_6|listname|string
```

`test-name` is the name of the test being performed, that is, *address*, *header*, *string*, and so forth. `owner` is the email address of the owner of the Sieve. `spare_4`, `spare_5`, and `spare_6` are the values of the corresponding SPARE_4, SPARE_5, and SPARE_6 attributes (set by the LDAP_SPARE_* MTA options) associated with the user LDAP entry the Sieve came from. Either owner or `spare_*` can be blank if the Sieve has no owner address or didn't come from an LDAP entry, respectively.

The intent of `spare_*` is to additional information about the location of a user's lists, for example, the `psRoot` LDAP attribute could be fetched in order to use personal address book entries as a source of list data. `listname` is the name of the list specified in the test itself. Finally, `string` is whatever string the

test is acting on, that is, it will be an address in the case of an address test, the contents of a header for a header test, or some string in the case of a string test.

The test succeeds if the mapping sets $Y. The string result of the mapping is interpreted as a series of properties separated by vertical bars. If variables are enabled the first property is stored in ${1}, the second in ${2}, and so on.

In addition to setting $Y and an appropriate result string if the test succeeds, entries in this mapping are required to perform an important piece of bookkeeping. If the test performed employs any of the `spare_*` fields, $* must also be set. $* tells the Sieve machinery that the test is recipient-specific and the script must be re-evaluated for each recipient. Failure to set $* can lead to botched test results for multi-recipient messages.

Redirect `:list` works differently. The probe format is basically the same except for the omission of the string parameter:

```
redirect|owner|spare_4|spare_5|spare_6|listname
```

However, the result returned is quite different. In the case of redirect in addition to setting $Y the mapping must return a URL which is then used as a source of addresses. A new MTA option, `MAX_REDIRECT_ADDRESSES`, limits the number of addresses will be read from such a list. It defaults to 128.

Redirect entries in the mapping also must set $* if they use `spare_*` information.

The `spare_*` fields are not set for system sieves prior to the Messaging Server 7 Update 1 release.

Example usage of extlists extension

The following example adds the header "X-Sender-in-PAB: yes" to an email if the sender address matches an email address in the users UWC/Communications Express/Convergence Personal Address Book.

- per-user Sieve Rule

```
require ["extlists","editheader"];
if (address :list "from" "checkPAB") {
     addheader "X-Sender-in-PAB" "yes";
}
```

- mapping table addition

```
SIEVE_EXTLISTS

! test-name|owner|spare_4|spare_5|spare_6|listname|string
! User doesn't have psRoot set
  address||*|*|*|checkPAB|*    $N
! User has psRoot set, perform a search
  address|*|*|*|*|checkPAB|*
$C$]$1??one?(|(piEmail1=$4)(piEmail2=$4)(piEmail3=$4))[$Y$*$E
```

- option.dat setting

```
LDAP_SPARE_4=psroot
```

- Additional ACI added to UWC/CE/Convergence PAB base to allow MTA read access

```
# Provide the MTA access to the user-PAB
dn: o=piServerDb
changetype: modify
add: aci
aci: (target="ldap:///o=piServerDb")
 (targetattr="*")
 (version 3.0; acl "PAB Administrator read rights"; allow
(read,search)
 groupdn="ldap:///cn=Messaging End User Administrators Group,
ou=Groups, o=isp";)
```

Sieve actions `setnotify` and `setreturn` added

Two new nonstandard Sieve actions have been added: `setnotify` and `setreturn`. These actions can only be used in system sieves. `setnotify` specifies a new value for the DSN NOTIFY parameter and `setreturn` specified a new value for the DSN RET parameter. Both of these parameters are specified in RFC 3461, as are the possible values that can specified for each one.

The primary use of these actions is expected to be to adjust return policies for suspected spam. For example, if address validation cannot be performed, it may be prudent to disable non-delivery reports, return of content, or both for messages suspected of being spam:

```
setnotify "NEVER";
setreturn "HDRS";
```

Example: Disable non-delivery reports, return of content and vacation auto-replies for spam email

```
<tcp_local channel filter>
require ["spamtest","relational","comparator-i;ascii-numeric"];
# Disable notifications and vacation messages for spam email
if spamtest :value "ge" :comparator "i;ascii-numeric" "5" {
    setnotify "NEVER";
    setreturn "HDRS";
    novacation;
}
```

Features Introduced in Messaging Server 7 Update 2

Features Introduced in Messaging Server 7 Update 2

New `streaming` channel keyword to control pipelining behaviour

A `streaming` channel keyword value of -1 or -2 now disables client use of pipelining completely even if the remote SMTP server offers it. -2 additionally prevents the SMTP server from announcing that pipelining is available.

New settings for `authrewrite` channel keyword

The `authrewrite` channel keyword has a new setting. Bit 6 (value 64), if set, causes a rewritten version of the envelope from address to be used in the probe as opposed to the original form. The specific rewritten form used is controlled by bit 7 (value 128): If set the canonical form return address will be used, if clear the normally rewritten form will be used instead.

Changed default setting for `ident*` channel keyword

The default `ident*` setting for all channels has been changed from `identnone` to `identnonelimited`. This is so channel switching based on DNS lookups is no longer done by default.

New setting for USE_ORIG_RETURN, USE_CANONICAL_RETURN, and USE_AUTH_RETURN MTA options

Bit 21 (value 2097152) of the USE_ORIG_RETURN, USE_CANONICAL_RETURN, and USE_AUTH_RETURN MTA options has been defined to control what return address is used in

FROM_ACCESS mapping probes. Since the FROM_ACCESS mapping was previously defined to use the original return address rather than a rewritten version, the default value of USE_ORIG_RETURN has been changed from 0 to 2097152 to preserve this behavior.

New mapping table metacharacter ($+x) to set flags

The $+x metacharacter sequence, where x is any letter, can now be used to set flags in mappings. In most cases $+x is equivalent to $x, except that $+x works for the letters C, E, L, P, R, and W, whereas the
metacharacters $C, $E, and so on are all used internally by the mapping machinery and cannot be used to set the corresponding flag.

The $-x metacharacter sequence, where x is any letter, can now be used to clear the corresponding flag.

New mapping table metacharacter ($+E) to set global sieve environment items.

$+E can now be used in a FROM_ACCESS, SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, or ORIG_MAIL_ACCESS mapping to set global sieve environment items. $+E causes two arguments are read from the mapping result string. The first specifies the name of the environment item and the second specifies the value the item should have. In the case of FROM_ACCESS these arguments are read immediately after any override postmaster value (enabled by $(or $)) and before any $+R value (see the next item). For all of the other mappings the arguments are read immediately after any `spamtest` adjustment argument (enabled by $,) and before any failure reason (enabled by $N).

Example: Adds the header "Auth-Sender: <authenticated email address>" for authenticated email uploads.

```
<mapping table entry>
FROM_ACCESS
!
port-access-probe-info|app-info|submit-type|src-channel|from-address|auth-t
Sets the "authsender" sieve environment (RFC 5183) parameter
! to be the authenticated sender address
  *|SMTP*|*|tcp_auth|*|*           $Y$+Eauthsender|$4

<tcp_auth channel filter>
require ["editheader","variables","environment"];

! Adds the header "Auth-Sender: <authenticated email address>"
if environment :matches "authsender" "*" {
        addheader "Auth-Sender" "${1}";
}
```

New mapping table metacharacter ($+R) to opt in to spam filtering

$+R can now be used in a FROM_ACCESS, SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, or ORIG_MAIL_ACCESS mapping to opt in to spam filtering. $+R causes two arguments to be read from the mapping result string. The first of these is an integer from 1 to 8 specifying which spam filter to activate. The second is the optin string to pass to the spam filter. In the case of FROM_ACCESS these arguments are read immediately after any environment items (controlled by $+E). In the case of the other mappings the arguments are read immediately after any additional Sieve URLs (enabled by $S) and only if $N/$F is not set.

New mapping table metacharacter ($*) to force disconnect of SMTP session

$* can now be used in conjunction with $N in a MAIL_ACCESS, SEND_ACCESS, ORIG_MAIL_ACCESS, or ORIG_SEND_ACCESS to force a disconnect of the SMTP session. Note that

this aborts the transaction and so should be used with great care.

Example:

```
ORIG_MAIL_ACCESS
! Immediately disconnect the connection if the source IP address is
listed in the zen.spamhaus.org
! blacklist
  TCP|*|25|*|*|*|*|tcp_local|*|*|* \
$*$C$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.]$E
```

$* can also be used in AUTH_REWRITE mappings to force a disconnect after processing of the current message concludes.

New FORWARD mapping table metacharacter ($/)

$/ in a FORWARD mapping in conjunction with YD will cause the resulting aliasing operation to be treated as a mailing list with the original envelope from as the error reporting address. This is similar in functionality to the specification of a / as a mgrpErrorsTo value or a / as the value of a nonpositional [envelope_from] parameter in either the alias file or database.

New INCLUDE_SPARES MTA option

LDAP attribute values associated with originator or recipient address processing may now be included in FROM_ACCESS or the various recipient address access mapping probes, respectively. This is all controlled by the INCLUDE_SPARES MTA option. This option is bit-encoded, with the bits defined as follows:

bit	value	meaning
0	1	Include LDAP_SPARE_1 attribute in FROM_ACCESS
1	2	Include LDAP_SPARE_2 attribute in FROM_ACCESS
2	4	Include LDAP_SPARE_3 attribute in FROM_ACCESS
3	8	Include LDAP_SPARE_4 attribute in FROM_ACCESS
4	16	Include LDAP_SPARE_5 attribute in FROM_ACCESS
5	32	Include LDAP_SPARE_6 attribute in FROM_ACCESS
6	64	Include LDAP_SPARE_1 attribute in ORIG_SEND_ACCESS
7	128	Include LDAP_SPARE_2 attribute in ORIG_SEND_ACCESS
8	256	Include LDAP_SPARE_3 attribute in ORIG_SEND_ACCESS
9	512	Include LDAP_SPARE_4 attribute in ORIG_SEND_ACCESS
10	1024	Include LDAP_SPARE_5 attribute in ORIG_SEND_ACCESS
11	2048	Include LDAP_SPARE_6 attribute in ORIG_SEND_ACCESS
12	4096	Include LDAP_SPARE_1 attribute in SEND_ACCESS
13	8192	Include LDAP_SPARE_2 attribute in SEND_ACCESS
14	16384	Include LDAP_SPARE_3 attribute in SEND_ACCESS
15	32768	Include LDAP_SPARE_4 attribute in SEND_ACCESS
16	65536	Include LDAP_SPARE_5 attribute in SEND_ACCESS
17	131072	Include LDAP_SPARE_6 attribute in SEND_ACCESS
18	262144	Include LDAP_SPARE_1 attribute in ORIG_MAIL_ACCESS
19	524288	Include LDAP_SPARE_2 attribute in ORIG_MAIL_ACCESS
20	1048576	Include LDAP_SPARE_3 attribute in ORIG_MAIL_ACCESS
21	2097152	Include LDAP_SPARE_4 attribute in ORIG_MAIL_ACCESS
22	4194304	Include LDAP_SPARE_5 attribute in ORIG_MAIL_ACCESS
23	8388608	Include LDAP_SPARE_6 attribute in ORIG_MAIL_ACCESS
24	16777216	Include LDAP_SPARE_1 attribute in MAIL_ACCESS
25	33554432	Include LDAP_SPARE_2 attribute in MAIL_ACCESS
26	67108864	Include LDAP_SPARE_3 attribute in MAIL_ACCESS
27	134217728	Include LDAP_SPARE_4 attribute in MAIL_ACCESS
28	268435456	Include LDAP_SPARE_5 attribute in MAIL_ACCESS
29	536870912	Include LDAP_SPARE_6 attribute in MAIL_ACCESS

Note that spare attribute slots can be assigned to attributes used for other purposes by the MTA.

New MTA options SPARE_n_SEPARATOR (n=1-6)

These options now control how multiple attributes that end up in a single spare slot are handled. These

options accept a non-negative integer value. The lower 8 bits of this value are interpreted as follows:

value	meaning
3	Multiple attribute values are not allowed - the user entry is considered invalid and ignored if multiples are present.
2	Use language tag information to decide which of the multiple attributes to use.
1	Multiple attributes are not allowed - the user entry is considered invalid if multiples are present.
0	Pick one of the values at random and use it.
32-255	Concatenate multiple attribute values together, using the character corresponding to the SPARE_n_SEPARATOR value as the separator.

The remaining bits of the option are used as bit flags. Currently only bit 8 (value 256) is defined: If set, it causes the attribute name and an equals sign to be prepended to the stored value.

The default values for SPARE_n_SEPARATOR are chosen to remain backwards compatible with previous behavior:

SPARE_1_SEPARATOR and SPARE_2_SEPARATOR: 1
SPARE_3_SEPARATOR: 0
SPARE_4_SEPARATOR, SPARE_5_SEPARATOR, SPARE_6_SEPARATOR: 2

Spare attribute slots increased from 6 to 18

The number of spare attribute slots has been increased from 6 to 18. This in turn creates new MTA options SPARE_n_SEPARATOR for n=7-18. Information from these slots is now placed in queue files so it can be accessed during dequeue processing. All of these new slots have a default SPARE_n_SEPARATOR value of 0, that is, if there are multiple attributes in a single slot, one will be picked at random.

New "spareN" alias nonpositional parameter

Nonpositional alias parameters of the form:

```
[spareN] value
```

where N=1-18 are now supported as a means of setting spare attribute values for addresses in the aliases file and database.

MTA support for external LDAP server lookups

Support has been added to the MTA to access an additional, external directory server. Special LDAP URLs with the prefix extldap: are used to specify access to this server. The following MTA options have been added to support this capability:

Option	Meaning	Default
LDAP_EXT_HOST	Host name of LDAP server.	""
LDAP_EXT_USERNAME	DN used to authenticate to this server. If no DN is specified authentication will not be used.	""
LDAP_EXT_PASSWORD	Password used to authenticate to this server.	""
LDAP_EXT_PORT	Port the LDAP server runs on.	389
LDAP_EXT_MAX_CONNECTIONS	Limit on the maximum number of connections that will be made.	(unlimited)
LDAP_EXT_INITIAL_CONNECTIONS	Number of connections to initially set up in the connection pool. A value of 0 results in connections only being created on an as-needed basis.	0

Sieve "ihave" extension support

The Sieve ihave extension, described in:

http://tools.ietf.org/id/draft-freed-sieve-ihave

is now supported. There are no configuration options associated with this support.

Sieve envelope-dsn and redirect-dsn extensions support

The Sieve envelope-dsn and redirect-dsn extensions, described in:

http://tools.ietf.org/id/draft-freed-sieve-notary

are now supported. There are no configuration options associated with this support.

Sieve mime extension support

The Sieve mime extension, described in

http://tools.ietf.org/id/draft-ietf-sieve-mime-loop-08.txt

has been implemented. Note that the other extensions described in this document have NOT been implemented.

The following example shows how to use the Sieve mime extension.

How Do I Block Emails with Executable File Attachments?

Messaging Server provides several mechanisms to block emails containing undesirable file attachments including the conversion channel, the milter-client interface, and integration with anti-virus/anti-spam engines. This FAQ provides instructions on how to use the more efficient Sieve filtering mechanism to block emails based on the extension of any files contained within that email.

> Note: This example does not completely solve the problem. Not all parts have content-disposition headers. You also need to check the actual media type as well as content-type name parameters.

1. Create the channel-level sieve filter file *msg_base*`/config/execblock.sieve` containing the

following information:

```
require ["mime", "refuse", "variables", "extlists"];
if header :mime :anychild :param "filename" :list
"Content-Disposition" "extension-check" {
 refuse "Message blocked due to ${0} attachement";
 stop;
}
```

2. Make sure the above file has the same owner, group and permissions as the other config files.
3. Enable this filter by specifying the above file as either sourcefilter or destinationfilter on the appropriate channel(s). For example, to have it applied to messages coming **from** the `tcp_local` channel, adding the following to the `tcp_local` channel entry in the *msg_base* `config/imta.cnf` MTA configuration file:

```
sourcefilter file:IMTA_TABLE:execblock.sieve
```

For example:

```
!
! tcp_local
tcp_local smtp mx single_sys remotehost inner switchchannel
identnonenumeric subdirs 20 maxjobs 7 \
pool SMTP_POOL maytlsserver maysaslserver saslswitchchannel
tcp_auth missingrecipientpolicy 0 \
loopcheck sourcefilter file:IMTA_TABLE:execblock.sieve
tcp-daemon
```

To also check internal emails, including emails sent by using webmail (`tcp_intranet`) or emails from authenticated senders (`tcp_auth`), then also add it to those channels.

4. Add the following mapping tables to your *msg_base*`/config/mappings` MTA configuration file:

```
SIEVE_EXTLISTS

 header|*|*|*|*|extension-check|*.* $C$|EXTLISTS;$5|$Y$E
 * $N

EXTLISTS

  exe   Blocked$ due$ to$ .exe$ attachment$Y
  bat   Blocked$ due$ to$ .bat$ attachment$Y
  bin   Blocked$ due$ to$ .bin$ attachment$Y
  *     $N
```

You can easily extend the preceding mapping table to block any other undesirable attachments, for example, "*.js" files.

5. Rebuild the MTA configuration files and restart the dispatcher:

```
# imsimta cnbuild
# imsimta restart dispatcher
```

6. Verify that the configuration is working by sending an email containing an executable attachment (<somefile>.exe). The email should be rejected with the following message:

```
550 5.7.1 Message blocked due to <attachment-filename> attachement.
```

7. If you want it to report the message from the EXTLISTS table instead of above, change the "refuse" line in the sieve filter to:

```
refuse "${1}";
```

Sieve capture action now supports Microsoft Exchange's "envelope journaling" format

Support has been added to the capture sieve action to produce Microsoft Exchange's "envelope journaling" format. This format consists of a multipart MIME message where the first part contains envelope information is a semi-structured format and the second part is the actual message. This new format by specifying a :journal parameter to capture:

```
capture :journal "trigger-address";
```

Body Extension Implementation

The restrictions on the support of RFC 5173 are:

1. The only match types supported are :contains and :is; :matches and :regex are not supported. This is likely to be a permanent restriction due to the possible performance impact of supporting these match-types.
2. The only body transforms supported are :raw and :text; :content is not supported. This restriction may be be lifted in a future release.
3. Variable substitutions are not allowed in body test arguments. If they are used an error is likely to occur. For example, this script will fail:

```
require ["variables", "body"];
set "a" "testing"
if body :contains "${a}" { discard; }
```

   This restriction exists so that a list of all arguments to body in all scripts can be computed in advance and searched for in a single pass. If this restriction were to be lift it's easy to construct scripts that require an arbitrary number of passes over the message, which is unacceptable in a server environment. As such, this should be considered to be a permanent restriction.

4. The :text body transform operates on all message parts with a text type or a 7bit/8bit encoding. If a charset other than utf-8 is specified on a text part that part is converted to utf-8 before being searched.

The availability of the body test is controlled by the `enable_sieve_body` MTA option. A value of 0, the default, disables the extension. A value of 1 enables the extension for use in all sieves. A value of 2 enables the use of body in system-level sieves only.

Additional switches for `imsimta test -rewrite` command

Additional switches have been added to `imsimta test -rewrite` command:

```
-spares - Show any spare attributes associated with the recipient
address
-soptin - Show any optins associated with the recipient address
-extra_local_channel=channel - Tell the rewriting machinery that
"channel" should have local channel
semantics. This is useful in testing rewriting associated with process
and conversion channel hops.
```

Additional verification functionality added to `imsimta test -domain` command

The verify command in `imsimta test -domain` now reports error if an underlying LDAP operation fails.

Additional {optin} macro for milter plugin

The milter interface now receives the spam filter optin value at the RCPT TO processing stage as an {optin} macro.

New `USE_JETTISON` option for milter plugin

A new option has been added to the milter plugin: USE_JETTISON. If this option is set to 1 sieve jettison will be used instead of discard in the sieves if the milter calls for the message to be discarded. A value of 0, the default, causes discard to be used.

> **Related Change Request**
> RFE #6802162 - "Provide option to translate milter server Discard response to jettison action"

MTA support for RFC 2231 encoded content-type and content-disposition parameters

Support has been added to the MTA to automatically desegment/decode and resegment/encode MIME content-type and content-disposition parameters. This now happens automatically - there are no configuration options associated with this enhancement.

> **Related Change Request**
> RFE #6570574 - "RFC 2231 support in the MTA"

Enhancement to charset conversions mechanism

Charset conversions now have the ability to remove content-disposition header fields. This is done by setting the out-disposition to the empty string. For example, an entry that would remove any content-disposition header from text/calendar parts would be something like:

```
in-channel=*; in-type=text; in-subtype=calendar; relabel=1;
out-disposition=
```

This particular action has become a useful thing to do because some Outlock/Exchange version combination refuse to process text/calendar parts with any sort of content-disposition header.

MTA message size calculations changed for enqueue log entries

The mechanism for computing size values in enqueue entries in the MTA transaction logs has been revised. Previously message sizes were computed based on counting octets in the input, which did not take various things, including charset-conversion, into account. It was done this way in order to facilitate certain calculations nedeed for performing message fragmentation. Now that message fragmentation has become a rarity, this approach is no longer appropriate, and the code has been changed to work directly with the output message.

Localization enhancement for `return_header.opt` file

The `return_header.opt` file is used to control what headers, if any, are included in the first part of delivery status notifications (DSNs), or message delivery notifications (MDNs) generated by the MTA. This file was always located in IMTA_TABLE: and hence was not localizable through the $I mechanism of the NOTIFICATION_LANGUAGE and DISPOSITION_LANGUAGE mappings. This has now been changed so that if $I is used the MTA will first look in the locale-specific location given by the mapping. If the file does not exist in that location the code will fall back to looking in IMTA_TABLE: instead.

Using MIMEDefang with Messaging Server

Using MIMEDefang with Messaging Server

MIMEDefang is a third-party milter server that you can configure to perform modifications to either the entire email body or individual email attachments, or both.

Messaging Server is able to leverage the flexibility of MIMEDefang to perform commonly requested email modifications through the use of the Messaging Server milter client plugin. This information provides general MIMEDefang information to assist with the installation and configuration of MIMEDefang plus several useful MIMEDefang modification examples.

Topics:

MIMEDefang Requirements

- Messaging Server 6.3 (starting with patch -20).
  You should use at least Messaging Server 7.0 Update 1 to make use of improved sieve handling functionality.
- MIMEDefang server software.
  You should use at least MIMEDefang 2.67.

Generic Hints and Tips for Using MIMEDefang

- Overview of Messaging Server and milter functionality
- Download location of MIMEDefang software
- Guide for installing MIMEDefang on Oracle Solaris
- Explanation of the mimedefang-filter configuration file

- Verify the syntax of your `mimedefang-filter` file by using `perl -c`, for example:

```
bash-2.05# perl -c /etc/mail/mimedefang-filter
/etc/mail/mimedefang-filter syntax OK
```

- Perl modules can be installed by using the CPAN mechanism, for example:

```
perl -MCPAN -e shell
```

- A sample MIMEDefang start-up script is provided in the following location:

```
<extract location>/mimedefang-<version>/examples/init-script
```

- To configure MIMEDefang to start on an TCP/IP port (versus a Unix Socket), edit the following parameter in the `init-script` to the following:

```
SOCKET=inet:<TCP/IP port number>
```

MIMEDefang Modifications Examples

Replace winmail.dat TNEF Attachments with Encoded Contents

Emails sent with Microsoft Outlook or Microsoft Exchange sometimes contain a single attachment with a **`winmail.dat`** file name. This attachment contains email formatting information and might also contain other files that the sender has attached to the email.

The `winmail.dat` file is encoded in the Microsoft Proprietary TNEF format. The vast majority of non-Microsoft third-party IMAP email clients (including Microsoft Outlook Express) and web-based mail clients (including Messenger Express, Communications Express, and Convergence) are unable to decode the TNEF encoded `winmail.dat` attachment and therefore cannot extract any files that the sender might have included in the email.

The following example MIMEDefang filter configuration is designed to extract any file attachments contained within the `winmail.dat` file and append these extracted files as a standard MIME base64 encoded attachment to the email. This greatly improves the chances that the recipient of the email is able to access the files that the sender attached to the email.

mimedefang-filter

```
sub filter {
    my($entity, $fname, $ext, $type) = @_;

    ### Convert TNEF winmail.dat format
    ### Note: You must install Convert::TNEF and File::Type from CPAN before
using this script
    if (lc($type) eq "application/ms-tnef" or lc($fname) eq "winmail.dat" ) {
        use Convert::TNEF;
        use File::Type;
        use File::Temp qw(tempfile tempdir);

        # Create a unique temporary directory under "/tmp"
        my $tnefdir = tempdir(CLEANUP => 1, DIR => "/tmp");
        if (not $tnefdir) {
                md_graphdefang_log('tnef_fail',"Unable to create temporary
directory");
                return action_accept();
        }

        # If we can't Convert the TNEF file for some reason, just accept the
attachment and log the error
        my $tnef = Convert::TNEF->read_ent($entity,{output_dir=>"$tnefdir"});
        if (not $tnef) {
                md_graphdefang_log('tnef_fail',$Convert::TNEF::errstr);
                return action_accept();
        }

        my $ft = File::Type->new();

        # Append attachments contained in the winmail.dat file to the
message.
```

```
        for ($tnef->attachments) {
             # Determine the mime-type of the file
             my $mimetype = $ft->mime_type($_->data);

                # File::Type doesn't detect text files well, this is a
workaround
             if ($mimetype eq "application/octet-stream") {
                  #Set the mime-type to text/plain if the first 1024
characters are printable
                  $text_check = substr($_->data,0,1024);
                  $mimetype = "text/plain" unless $text_check =~
/[^[:print:]\s]/;
             }

             my $tnef_entity = action_add_part($entity, "$mimetype",
"base64", $_->data, $_->longname, "attachment");
             md_graphdefang_log('tnef_ext', "File: " . $_->longname . " Type:
$mimetype");

             # Run each new TNEF-sourced MIME part back through the filter
again, this ensures that bad filenames etc.
             # cannot sneak through by being contained in winmail.dat files

             filter ($tnef_entity, $_->longname, "", "$mimetype");
        }

        # Deletes working files
        $tnef->purge;

        # Remark this if you want still want to keep the original winmail.dat
file
        return action_drop();
    }

    # Keep the attachment
```

```
    return action_accept();
}
```

Append Footer/Disclaimer to Email

You use `append_text_boilerplate` and `append_html_boilerplate` MIMEDefang functions to append an additional message to the bottom of an email, whether it be an email disclaimer for legal reasons or a common organizational footer.

The following example shows how to use these functions to add a footer to the bottom of plain-text and HTML parts.

mimedefang-filter

```
sub filter_end {
    my($entity) = @_;

    my $text_footer = "------------------------\n" .
                "[Insert Standard Plain-Text Disclaimer Here]";

    my $html_footer = "------------------------<br/>" .
                "[Insert Standard HTML format Disclaimer Here]";

    ## Uncomment the following line to append the standard-disclaimer to the
first plain-text part
    #append_text_boilerplate($entity, $text_footer, 0);

    ## Uncomment the following line to append the standard-disclaimer to the
first html-text part of the email
    ## Note: You must install HTML::Parser from CPAN before using
append_html_boilerplate
    #append_html_boilerplate($entity, $html_footer, 0);
}
```

Block, Replace, or Rename Executable Attachments

A common pro-active measure to prevent virus infected attachments that were not detected by virus scanners as infected is to block, replace, or rename "executable" attachments. This technique reduces the chance that the user will attempt to execute the attachment and infect the system.

The following example provides the mechanisms to block, replace, or rename an executable attachment.

mimedefang-filter

```
## Override the default notification advising that MIMEDefang has modified
the email
$GeneralWarning =
   "NOTICE: This email has been modified.\n" .
   "Please contact your system administrator for further details.\n" .
   "The reason for the modification is provided in the following
paragraph:\n\n";

# This procedure returns true for entities with bad filenames.
```

```
sub filter_bad_filename  {
    my($entity) = @_;
    my($bad_exts, $re);

    # Bad extensions
    $bad_exts =
'(ade|adp|app|asd|asf|asx|bas|bat|chm|cmd|com|cpl|crt|dll|exe|fxp|hlp|hta|hto|i
MIMEDefang with Messaging Server^\}]+\})';

    # Do not allow:
    # - CLSIDs  {foobarbaz}
    # - bad extensions (possibly with trailing dots) at end
    $re = '\.' . $bad_exts . '\.*$';

    return 1 if (re_match($entity, $re));

    # Look inside ZIP files
    if (re_match($entity, '\.zip$') and
        $Features{"Archive::Zip"}) {
        my $bh = $entity->bodyhandle();
        if (defined($bh)) {
            my $path = $bh->path();
            if (defined($path)) {
                return re_match_in_zip_directory($path, $re);
            }
        }
    }
    return 0;
}

## Warning message added when attachments are renamed by the action_defang
method
sub defang_warning {
    my($oldfname, $fname) = @_;
    return "The attachment '$oldfname' was renamed to '$fname'\n";
}

sub filter {
    my($entity, $fname, $ext, $type) = @_;

    ## Action attachments with "bad" filenames.
    if (filter_bad_filename($entity)) {
        md_graphdefang_log('bad_filename', $fname, $type);

        ## Uncomment the following three lines to drop the attachment and
replace it with a warning
        # return action_drop_with_warning("An attachment named $fname was
removed from this document as it\n" .
        #          "constituted a security hazard.  If you require this
document, please contact\n" .
        #          "the sender and arrange an alternate means of receiving
it.\n");

        ## Uncomment the following line to rename the attachment to a
auto-generated replacement
        # return action_defang($entity, "", "", "application/octet-stream")

        ## Uncomment the following line to block the email at the SMTP level
        ## Note: This functionality works best with Messaging Server 7.0
```

```
update 1 and above.
        # return action_bounce("Executable attachment $fname banned due to
site policy, email rejected");
```

```
    }
}
```

Features Introduced in Messaging Server 7 Update 3

Features Introduced in Messaging Server 7 Update 3

New Default Values for USE_REVERSE_DATABASE and USE_DOMAIN_DATABASE

The default value the MTA gives to the `USE_REVERSE_DATABASE` MTA option when no setting is present in `option.dat` has been changed from 5 to 4. Similarly, the default for `use_domain_database` has changed from 1 to 0.

`mail.log` Entries Include `spamtest` and `virustest` Levels

The spamtest and virustest levels in effect for the active sieve for a given recipient will now be included in E `mail.log` entries when LOG_FILTER is enabled. This will appear between the sieve name and the action list. For example:

```
[file:///blahblahblah], spamtest 26.000000, discard
```

New Form of Domain Literal

A new form of domain literal is now provided to explicitly route to a channel by name without requiring any rewrite rules. The new
domain literal format is:

`[channel:`*name*`]`

where *name* is the channel name. When such a domain literal is used, a check for a rewrite is made. If no such rule exists and the named channel is part of the configuration, a rule is synthesized of the general form:

`[channel:`*name*`]  $U%$D@`*official-host-name*

where *official-host-name* is the official channel host associated with the channel *name*.

The primary use of such literals is expected to be in source routes, for example:

`@[channel:`*process*`]:`*user@domain*

Nonstandard `addprefix` and `addsuffix` Sieve Actions Added

These actions each take a single string argument. The content of that argument is prepended or appended, respectively to the first plain text part found in the message using the same facilities employed by the `mgrpMsgPrefixText` or `mgrpMsgSuffixText` group attributes or the `[prefix_text]` and `[suffix_text]` alias file nonpositional parameters. Note, however, that these actions work in any sort of Sieve, not just Sieves attached to groups. If multiple Sieves are active and more than one prefix or suffix is specified, they are concatenated.

Changes to the `mailDeferProcessing` LDAP Attribute

The `mailDeferProcessing` LDAP attribute now accepts, in addition to the values `Yes`, `No`, and `After_auth`, MTA channel names. If a channel name is specified, alias expansion is deferred and the message is forcibly routed to that channel using a source route of the general form:

`@[channel:`*name*`]:`

The attribute is ignored if the named channel does not exist. Note that it only makes sense to forcibly route to some sort of process or reprocess channel.

Additional Enhancements to the `authrewrite` Channel Keyword

Additional enhancements have been made to the `authrewrite` channel keyword and `AUTH_REWRITE` mapping. First, bit 8 (value 256) of `authrewrite`, if set, causes the `AUTH` parameter from the `MAIL FROM` command to be included in the probe just after the `auth-sender` field.

Second, if the mapping sets `$O`, an additional result will be read from the mapping output string after reading any headers added with `$A` but before reading any error information associated with `$N`. This result replaces the current `AUTH` parameter from the `MAIL FROM` command.

New Channel Options `sasltrustauth` and `nosasltrustauth`

If `sasltrustauth` is set on a source channel, any value presented in the `MAIL FROM AUTH` parameter will be promoted to the authenticated originator address used throughout the MTA. `nosasltrustauth` is the default. The `sasltrustauth` option should be used with great care because the `AUTH` parameter is not, in general, trustworthy and the authenticated originator address is used for a variety of authentication checks.

Support for the `envelope-auth` Sieve Extension

Support for the `envelope-auth` Sieve extension has been implemented. This extension adds one additional envelope-part string: "auth", which provides access to any `AUTH` value specified on the SMTP `MAIL FROM` command for the current transaction.

New Sieve Environment Items Added

Two new Sieve environment items, `vnd.sun.authenticated-sender-address` and `vnd.sun.authenticated-sender-id`, have been added. The former provides access to the sender address associated with the authentication state for the SMTP session. The latter provides similar access to the user identity.

Status Setting of `deliver` for `mailUserStatus`

A `mailUserStatus` of `deliver` is now treated by the MTA as equivalent to `active`. This new status settings allows mail delivery to continue uninterrupted while effectively locking the user out their account (because other components will treat `deliver` the same way as `inactive`).

Change in Policy for Missing Recipient Header Fields

The policy for missing recipient header fields for `tcp_auth`, `tcp_submit` and `tcp_intranet` in the default configuration has been changed from "add empty group" to "don't do anything."

Change in Default Number of Subdirectories in Channel Queues

The default for the number of subdirectories created from channel queues has been changed from -1 (no subdirectories) to 20.

Change in Default for `*sendpost` and `*warnpost`

The default for `*sendpost` and `*warnpost` has been changed from `copy*post` to `no*post`.

`copysendpost` and `copywarnpost` Setting Removed From Default Configuration

The setting of `copysendpost` and `copywarnpost` in the default configuration has been removed, meaning that `nosendpost` and `nowarnpost` are now the default.

Setting the SMTP Protocol Delay

$+*n*P and $-*n*P, *n* = 0 or *n* > 5, when used in any SMTP enqueue-related mapping, sets the protocol delay to *n* centiseconds. $+1P inserts the current protocol delay setting into the mapping output string.

Changes to the `SPAMFILTERX_FINAL` MTA options

Bit 2 (value 4) of any of the `SPAMFILTERX_FINAL` MTA options, if set, causes the "initial" address to be passed to the spam filter. This is the address that was initially passed to the alias expansion process. Bit 1 can be used to strip source routes from such addresses if desired.

`service` and `noservice` channel keywords replaced

The `service` and `noservice` channel keywords have been replaced by `serviceconversion` and `noserviceconversion`, respectively. The old keywords will continue to operate.

`mgrpJettisonDomain` and `mgrpJettisonBroadcasters` Added to MTA's Group Expansion Facility

Two new attributes, `mgrpJettisonDomain` and `mgrpJettisonBroadcasters`, have been added to the MTA's group expansion facility. `mgrpJettisonDomain` specifies one or more domains whose

messages should be jettisoned if sent to this group. A match marks the message to be jettisoned and bypasses all other group checks and expansion. Multiple attributes and multiple values are allowed, as are glob-style wildcards. The LDAP_JETTISON_DOMAIN MTA option may be used to specify a different attribute for this purpose if desired.
mgrpJettisonBroadcasters works in a similar way, but specifies a URL identifying mail addresses whose messages should be jettisoned if sent to this group. Multiple attributes and multiple values are again allowed. Each URL is expanded into a list of addresses and each address is checked against the current envelope from address. A match marks the message to be jettisoned and bypasses all other group checks and expansion. Substitution processing will be performed on this URL If bit 6 (value 64) of the PROCESS_SUBSTITUTIONS MTA option is set. The LDAP_JETTISON_URL MTA option may be used to specify attribute(s) other than, or in addition to, mgrpJettisonBroadcasters, with these semantics.

New MAX_SIEVE_STRING_SIZE MTA Option,

A new MTA option, MAX_SIEVE_STRING_SIZE, has been added. This option controls how large strings can be in Sieve scripts. It defaults to 65536.

New MILTER_MACROS Mapping

Support for a new MILTER_MACROS mapping has been added. MILTER_MACROS is called by the milter spam filter plugin each time a macro is passed to the milter server. The probe format is:

```
spamfilter-index|command|macroname|macrovalue
```

Here *spamfilter-index* is an integer between 1 and 8 specifying the spam filter slot this milter is in, *command* is the command this
macro precedes, one of:

CONNECT
MAIL
RCPT

macroname is simply the name of the macro being defined and *macrovalue* is its value.

When the mapping returns, if $N or $F are set the macro is dropped and never sent to the milter server. If neither $Y nor $T causes any result the mapping produced to be discarded in favor of using the original macro name and value.

If, however, $Y or $T is set, the mapping result is processed as a series of macro name/value pairs, each name or value separated by vertical bars.

Finally, if $| is set in addition to $Y or $T, only a single name-value pair is read from the result and the second and subsequent vertical bars are treated as part of the value.

New vnd.sun.autoreply-internal Sieve Environment Item

A new Sieve environment item, vnd.sun.autoreply-internal, has been added. This item returns TRUE when the autoreply criteria for using an internal autoreply response have been met. Otherwise, this item returns FALSE.

New disablesourcefilter and disabledestinationfilter Channel Keywords

These keywords can be used to suppress the evaluation and interpretation of Sieve filters based on source or destination channel respectively. Each keyword takes a single nonnegative integer argument, whose value is interpreted as follows:

Value	Description
0	DIsable all Sieves
1	Only spam filter sieves are evaluated and interpreted
2	Only spam filter and source channel sieves
3	Spam filter, source channel, and system sieves
4	Spam filter, source channel, system, and destination channel
5	Spam filter, source channel, system, destination channel, and ORIG_SEND_ACCESS sieves
6	Spam filter, source channel, system, destination channel, ORIG_SEND_ACCESS, and SEND_ACCESS sieves
7	Spam filter, source channel, system, destination channel, ORIG_SEND_ACCESS, SEND_ACCESS, and ORIG_MAIL_ACCESS sieves
8	Spam filter, source channel, system, destination channel, ORIG_SEND_ACCESS, SEND_ACCESS, ORIG_MAIL_ACCESS, and MAIL_ACCESS sieves

New `removeconversiontag` Sieve Action

This action can be used to undo all or part of a preceeding `addconversiontag` or `setconversiontag` operation. Like the other two actions, this action accepts a single argument specifying either a string or list of conversion tags, which in this case are to be removed from the current conversion tag set.

Change to Default `ALIAS_DOMAINS` MTA Option

The default value for the ALIAS_DOMAINS MTA option has been changed from 1 to 6.

Change to Default `ALIAS_MAGIC` MTA Option

The default value for the ALIAS_MAGIC MTA option has been changed from 98764321 to 8764.

`DIRSYNC_HACK` MTA Option Remove

The DIRSYNC_HACK MTA option has been removed.

Envelope Journaling Code Strips Source Routes from Recipient Addresses

The envelope journaling code has been modified to unconditionally strip source routes from recipient addresses.

Computing a Hash In a Mapping Template

$+*n*#...#, *n* > 0, in a mapping template computes a hash of the specified string and returns the result. Additional radix, width, and modulus arguments may be specified using the same conventions as in a sequence number callout. Currently only n = 1 is defined; the hash function applied in this case is:

```
int hashvalue(char *string, int length)
{
  unsigned int hash;
  int i, j;
  unsigned int *uptr;

  uptr = (unsigned int *)string;
  hash = length;
  j = length >> 2;
  for (i = 0; i < j; i++)
  {
    hash ^= *uptr++;
    hash = (hash << 9) | (hash >> 23);
  }
  for (i = j << 2; i < length; i++)
  {
    hash ^= string[i];
    hash = (hash << 13) | (hash >> 19);
  }
  return ((hash * 0x71279461U) >> 2);
}
```

New Nonpositional spare1 Through spare18 Alias Parameters Added

These correspond to the spare LDAP attribute slots and allow an entry in the alias file or database to set these values in a fashion
comparable to an user or group defined in LDAP.

Prefix and Suffix Text Additions

Prefix and suffix text additions will now occur on the first text part within a nested multipart structure. Multipart/alternative is excluded from such actions. Previously additions only occurred if there was only a single text part present.

Change to LANGDIR MTA Option

The LANGDIR MTA option can now be set to a list of directory paths separated by spaces or commas. Additionally, an override directory specified in a NOTIFICATION_LANGUAGE or DISPOSITION_LANGUAGE mapping can also be a comma-separated list. Each directory on the list is checked in order for a given return.txt or disposition.txt file until the file is located or the directory list is exhausted. Additionally, the default for the LANGDIR MTA option has been changed from:

```
IMTA_TABLE:locale/C/
```

to:

```
IMTA_TABLE:locale/C/,IMTA_LIB:locale/C/
```

As a result of this enhancement, the iMTA_LANG "tailor variable" (which in fact does not come from the tailor file at all - it is constructed from the LANGDIR setting) has lost most if not all of its value. It returns the first value from the LANGDIR list and no others. As such IMTA_LANG should be considered deprecated and will be removed in a future release.

Features Introduced in Messaging Server 7 Update 4

Features Introduced in Oracle Communications Messaging Exchange Server Version 7 Update 4

Topics:

`ignoremultipartencoding` and `ignoremessageending` Channel Options

These channel options now affect MIME processing done by the conversion, SMS, calendar, and user-written channels.

More Stringent Content Checking of the Mappings File

The MTA now checks the contents of the mappings file more stringently. In particular, failure to indent the lines inside a mapping will now result in an error. This will provide early detection of some common formatting problems.

Changes to the `discard` and `jettison` Sieve Actions

The `discard` and `jettison` Sieve actions now accept a single, optional string parameter. This parameter value, if specified, is logged as part of the filter result in the MTA tranaaction log, assuming of course that the associated sieve is the one that determines the disposition of the current message. Note that this argument is nonstandard, however, since the main application is in system-level sieves, this should not present a portability problem in practice.

Nonstandard `setpriority` Action Added to MTA's Sieve Implementation

A nonstandard `setpriority` action has been added to the MTA's Sieve implementation. THis action is only allowed in system-level sieves. A single argument is required: The effective priority used for processing the message. Possible values are: "non-urgent", "normal", and "urgent". Note that this priority is NOT stored in the message header and only affects processing at this particular stage of message transfer. If multiple `setpriority` actions are specified in different system-level sieves, the one in the most specific sieve wins.

New `disconnectcommandlimit` Channel Option

When placed on a source channel, it imposes a liimit on the total number of commands that can be issued in a single SMTP session.

Application of Default Charset Label to content-type and content-disposition Parameters

MIME content-type and content-disposition parameters that contain illegal 8bit material are forcibly encoded by the MTA. Support has now been added to apply whatever default charset label is in effect to such parameter values in addition to encoding them.

New MAX_PARTS Channel Option Added to the defragment Channel

The defragment channel now has a channel option: MAX_PARTS. MAX_PARTS determines the maximum number of parts a message can have and still be reassembled. The maximum is 100,000; the default is 1000.

Enhancements to the MTA's BURL Support

Three enhancements have been made to the MTA's BURL support:

- Setting $M in the BURL_ACCESS mapping overrides the host name in imap: URLs with the mailhost attribute of the currently authenticated user. The BURL command will fail if $M is set and there is no default mailhost.
- Setting $I in addition to $N in a BURL_ACCESS mapping causes the session to disconnect after the error response is issued.
- New channel option disconnectbadburllimit. This channel option takes a single integer parameter specifying the number of invalid BURL commands that will be allowed before disconnecting. The option defaults to 3.

Implementation of Sieve Date and Index Extensions Described in RFC 5260

The Sieve date and index extensions described in RFC 5260 have been implemented. (Note that prior to this release, certain parts of these extensions, notably the currentdate and parts of the date test, had already been made available.)

New JOURNAL_FORMAT MTA Option

Starting with Messaging Server 7 Update 4 Patch 19, a new JOURNAL_FORMAT MTA option has been added to control the format of Exchange journaling messages generated by the MTA. This is a bit-encoded option. Currently assigned bits are described in the following table:

Bit/Value	Description
Bit 0 (value 1)	If set, generates the basic 2007 journal format instead of the 2003 format.
Bit 1 (value 2)	If set, sets the From:/To:/Subject: of the journal message to be the same as the message being journaled. Note that setting this may cause looping problems for setups that use header checks to determine what messages to archive.
Bit 2 (value 4)	If set, generates a X-MS-Exchange-Organization-Journal-Report: header field rather than a X-MS-Journal-Report: field.
Bit 3 (value 8)	If set, includes expanded/forwarded address information in the report (if such information is available). Note that bit 0 must also be set for this to work.

The default value for this option is 0. This option is intended to facilitate interoperating with Exchange itself, in particular, so that MTA-generated journal messages can be imported into Exchange.

New AUTH_DEBUG SMTP Channel Option

{{AUTH_DEBUG can be set to a list of debug keys that in turn will be passed to the HULA subsystem. Additionally, setting $A in the PORT_ACCESS mapping table causes a set of debug flags to be taken from the mapping result and used. This happens before any other use of the mapping result.

Features Introduced in Messaging Server 7 Update 5

Features Introduced in Oracle Communications Messaging Server Version 7 Update 5

Messaging Server 7 Update 5 includes new features introduced in Messaging Server 7 Update 5 Patch 28 and Messaging Server 7 Update 5 Patch 29 (Messaging Server 7.0.5.29.0). The new features in Patch 29 are:

- Access to Intermediate Address Information in Forward Mapping
- New Destination-Channel Keyword for Reducing Message Headers to a Specified Number of Bytes
- New USE_REVERSE_DATABASE Bit Settings Allow Originator-Based Restrictions
- Logging Has Been Enhanced to Include Individual Conversion Tags in Filter Log Entries

The following is a complete list of the features introduced in Patch 28 and Patch 29.

Features:

- Access to Intermediate Address Information in Forward Mapping
- New Destination-Channel Keyword for Reducing Message Headers to a Specified Number of Bytes
- New USE_REVERSE_DATABASE Bit Settings Allow Originator-Based Restrictions
- Logging Has Been Enhanced to Include Individual Conversion Tags in Filter Log Entries
- Updates to the Sieve Address Test
- New MTA Option: osync
- MTA Rejects UIDs That Begin With a Hyphen (-)
- New MTA Option: ldap_group_dn2
- Enhanced Use of ldap_group_dn and ldap_group_dn2
- GROUP_AUTH Mapping and New MTA Options ldap_auth_mapping *N* (*N*=1-4) Added
- Maximum Length of mailAutoReplySubject LDAP Attribute Increased
- Lookup Failure String Set by the $. Metacharacter Sequence Enhanced
- Enhanced V Transaction Log Entries
- SMTP Client Trace Level Debugging Support
- Enhanced spamfilter *N*_received (*N*=1,8) MTA Options
- New MTA Option JOURNAL_FORMAT Controls Exchange Journaling Messages
- Notify Actions in System-level Sieves No Longer Canceled
- New Channel Options: noaddrtypescan, addrtypescan and addrtypescanbccdefault
- New Data File: tlds.txt
- New Rewrite Rule Metacharacters $, and $>
- internet.rules File Modified
- Handling and Default Values for Some Error Text Options Changed
- System-level Sieve Loop Constructs
- New Channel Options: sourceconversiontag and destinationconversiontag
- New Destination Channel Options: parameterformatdefault, parameterformatminimizeencoded, and parameterformatstripencoded
- New spamfilter MTA Options: spamfilter *N*_includeheaders (*N* = 1,8)
- New Channel Option: msexchange
- Nonstandard :aindex Tag With Sieve Address Test
- authrewrite and AUTH_REWRITE Enhancements
- New MTA Options: scan_channel, scan_origin, and scan_recipient
- New Sieve Environment Item: vnd.oracle.last-verdict
- Scan Mode Added to the imsimta test mime Utility
- New Tailor Variable: IMTA_HOST.
- New Sieve Address Test Part Modifiers: :display and :comment
- :raw and :text Modifiers Enhanced

- MTA Address Reversal Logic Redesigned to Improve Subaddress Handling
- MTA Folds Multiple To:, Cc:, and Bcc: Fields into a Single Field
- Support for RFC 5435, RFC 5436
- Renamed job_controller Options
- Some LDAP URL and DNs Errors Ignored
- imsimta test rewrite Supports size=n Qualifier
- Dispatcher Options Default Value Changes
- Messaging Server Initial Configuration Modified to Generate Correct INTERNAL_IP Mapping Table
- New MTA Option: LOG_AUTH
- iSchedule Channel Handles iMIP Messages
- New Mapping: AUTH_ACCESS
- New Delivery Option Modifier: %
- MAX_NOTIFYS and MAX_VACATIONS MTA Options Modified
- Mapping Template Quoting Restrictions Removed
- New Delivery Option: nomail
- LMTP Server Supports STARTTLS.
- Out-encoding Clauses Now Honored While Relabeling
- New In-encoding Clause Allowed in Conversion Files
- SMS Gateway Server RFC 3458 Support
- MTA's Sieve External List Support Modified
- importancetest Sieve Test and importanceadjust Sieve Action Implemented
- imsimta test rewrite Utility Supports identifier Switch
- New MTA Option: LDAP_CHECK_HEADER
- New imsimta calc mm and imsimta test expression mm switch: system
- Spaces Now Valid as Delimiters in Conversion Tag Strings
- SMS Gateway Server Can Deliver Content in Subject Header Line
- New MTA Options: LOG_CONVERSION_TAG, LOG_IMAP_FLAGS, and LOG_DELIVERY_FLAGS
- SMTP Server Rejects STARTTLS When Followed by Additional Commands
- New SMTP Channel Option: CLIENT_CERT_NICKNAME
- New Bits/Values Added to MTA Option: LDAP_USE_ASYNC
- New Bit/Value Added to MTA Options: USE_CANONICAL_RETURN and USE_ORIG_RETURN
- Setting the LOG_FORMAT MTA Option to 4 Now Causes Log Entries to Be Written in an XML-Compatible Format
- Support for the Vacation Seconds Extension
- Support Added for a Domain-Level Alias detour host optin Attribute
- Source Channel Based Alias Detour Processing Now Works on Alias File and Alias Database Expansions
- New Bit/Value Added for MTA Option: INCLUDE_CONVERSIONTAG
- New TCP Channel Option: SSL_CLIENT
- Facilities Provided for Selecting Different Capture Formats for Capture Requests Made in Mappings or Through LDAP
- RESETDEBUG Option Added to Archiving Spamfilter Plugin
- Bit/Value of MTA Option: LOG_MESSAGES_SYSLOG Sets syslog Priority and Facility Code
- IP_ACCESS Mapping Includes Channel the Message Is Being Dequeued From
- MMP configutil support And Configuration Simplification
- MMP Honors configutil Settings
- Support for IMAP STARTTLS When Proxies (MMP, mshttpd, IMAP shared folders) Connect to Back-end
- MMP Support for IMAP Response Codes (RFC 5530)
- MMP and IMAP Performance Improved on Solaris by Use of Solaris Event Ports
- MMP Scalability Improvements: mmp.numprocesses
- IMAP Authentication Messages Now Include Session ID
- Ability to Disable Remote POP Collection for Webmail Has Been Added
- Support For Third-Party-Authentication Co-Process
- The libibiff and libjmqnotify Notification Plugins Are Now Built-In to libstore
- Password Expiration and IMAP Expiration Warning ALERTs
- New C <channel> Option

- imexpire Enhancements
- imexpire by IMAP User Flags
- Mechanism to Schedule Cleanup Immediately
- ENS Events Include Quota Information

Access to Intermediate Address Information in Forward Mapping

Bit 7 (value 128) of `USE_FORWARD_DATABASE`, if set, now includes the initial address presented for alias processing in the `FORWARD` mapping probe. Bit 8 (value 256), if set, now includes the current intermediate address in the `FORWARD` mapping probe. These addresses appear immediately before the final recipient address.

New Destination-Channel Keyword for Reducing Message Headers to a Specified Number of Bytes

A new destination-channel keyword, `headercut,` has been added. If specified, `headercut` cuts the current message header down to no more than a specified number of bytes, using a heurestic algorithm that removes or truncates header fields based on their relative importance. `headercut` requires a single nonnegative integer argument. A value of 0, the default, disables header cutting.

New USE_REVERSE_DATABASE Bit Settings Allow Originator-Based Restrictions

Bits 13 (value 8192) and 14 (value 16384) in the `USE_REVERSE_DATABASE` MTA option have been defined so that, if set, they disable source block and recipient limit settings and capture actions based on the envelope from (MAIL FROM) address (bit 13) and authenticated sender address (bit 14).

Logging Has Been Enhanced to Include Individual Conversion Tags in Filter Log Entries

If the `LOG_FILTER` MTA option is set, a conversion tag added with either the `setconversiontag` or the `addconversiontag` Sieve action is now logged in `mail.log*`.

Updates to the Sieve Address Test

The Sieve address test now uses the heuristic address parser instead of the strict parser. This helps address tests work even when the header field contains one or more syntax errors.

New MTA Option: osync

If MTA option `osync` is set to `1`, it causes the `O_SYNC` flag to be set when creating queue entries. The default is `0`.

> **Note**
> Setting `O_SYNC` may provide an increase in performance on ZFS file systems but will degrade performance considerably on UFS file systems.

MTA Rejects UIDs That Begin With a Hyphen (-)

The MTA now rejects UIDs that begin with a hypen (-) because they conflict with IMAP ACL formats. The Message Store also continues to perform this validation.

New MTA Option: ldap_group_dn2

The ldap_group_dn2 MTA option defines a new attribute used to provide group expansion information. The semantics of this new attribute slot are the same as the existing ldap_group_dn MTA option. Like any other alias attribute slot, multiple attributes may be assigned to a single slot. However, any given group can employ at most one such attribute. The second slot is intended to be used for groups that are defined in terms of two different attributes.

Enhanced Use of ldap_group_dn and ldap_group_dn2

The attributes selected by the ldap_group_dn and ldap_group_dn2 are normally used to specify DNs. By choosing a different value for the group_dn_template MTA option, other expansion identifiers may be used.

For example, if ldap_group_dn is changed to memberid and the default value of group_dn_template is changed from:

```
ldap:///$A??sub?(mail=*)
```

to:

```
ldap:///<some-dn>??sub?(memberof=$A),
```

the group now contains all entries with a memberof attributes whose value matches the memberid attribute on the group. One limitation of this mechanism is that all such attributes must have the same semantics. This issue has been addressed by the addition of the group_template mapping. When a mapping with this name exists, it is probed each time such an attribute is expanded. The probe format is:

```
attribute-name|attribute-value
```

If the mapping sets $Y, then the mapping string result will be used as the template for this attribute instead of the GROUP_DN_TEMPLATE value. If $N is set the attribute will be silently ignored.

GROUP_AUTH Mapping and New MTA Options ldap_auth_mapping*N* (*N*=1-4) Added

The GROUP_AUTH mapping and three new MTA options ldap_auth_mapping*N* (*N*=1-4) have been added to facilitate a flexible processing model when existing group access control mechanisms cannot be modified for increased access and permissions.

The MTA options are used to define attribute names that are fetched during alias expansion processing. When the mapping is defined and at least one of the four attributes is defined and appears on a group, the mapping is probed during group authorization checks (before any other authorization checks are done).

The probe format is:

```
envelope-from|group-address|auth1|auth2|auth3|auth4
```

The auth*N* fields are the values associated with ldap_auth_mapping*N* attributes for the group. If multiple attributes or multiple attribute values appear, they will all be present, separated by commas. The mapping can produce any one of the following outputs:

- $Y indicates that the authentication check has been passed.
- $T indicates that the mapping result is a URL, which is then checked in the same fashion as an LDAP_AUTH_URL would be.
- $N indicates that authentication has failed.

- $F indicates that the mapping result is a URL, which is then checked in the same fashion as an LDAP_CANT_URL would be.

Maximum Length of mailAutoReplySubject LDAP Attribute Increased

The maximum length of a :subject parameter in a Sieve vacation action has always been 1024 octets.

The length of the corresponding mailAutoReplySubject LDAP attribute is now 750 octets, increased from 256. The actual limit varies depending both on what other auto reply attributes are used and how many characters in the subject line have to be quoted. For example, the limit could be as low as 375 octets if the string consists entirely of double quotes.

Lookup Failure String Set by the $. Metacharacter Sequence Enhanced

The lookup failure string set by the $. metacharacter sequence now applies to domain attributes (i.e., $}domain,attribute{) in addition to regular LDAP lookups.

Enhanced V Transaction Log Entries

V transaction log entries now contain a reason field specifying at what point in the transaction the failure occurred and the routine that logged the failure.

SMTP Client Trace Level Debugging Support

You can now enable trace level debugging (TCP/IP channel-specific option TRACE_LEVEL=2) for specific SMTP clients. To do this, specify $G in the PORT_ACCESS mapping entry used to match the client's IP address.

Enhanced spamfilter*N*_received (*N*=1,8) MTA Options

The spamfilter*N*_received (*N*=1,8) MTA options now accept a value in the range of 0-2.

- 0: don't add a synthetic Received: field to the message content passed to the spam filter
- 1: add a synthetic Received: field to the message content passed to the spam filter (default value)
- 2: add an additional clause of the form
  (envelope-sender <MAIL-FROM-ADDRESS>)
  to the header. This clause is used by some SpamAssassin configuration as a source of MAIL FROM addresses instead of using the standards-compliant Return-path: field.

New MTA Option JOURNAL_FORMAT Controls Exchange Journaling Messages

A new journal_format MTA option has been added to control the format of Exchange journaling messages generated by the MTA.

This is a bit-encoded option. Currently assigned bits are:

- Bit 0 (value 1) - If set, generate the basic 2007 journal format instead of the 2003 format.
- Bit 1 (value 2) - If set, set the From:/To:/Subject: of the journal message to be the same as the message being journalled. Note that setting this may cause looping problems for setups that use header checks to determine what messages to archive.
- Bit 2 (value 4) - If set, generate a X-MS-Exchange-Organization-Journal-Report: header field rather than a X-MS-Journal-Report: field.
- Bit 3 (value 8) - If set, include expanded/forwarded address information in the report (if such information is available).

> **Note**
> Bit `0` must also be set for this to work.

The default value for this option is `0`. This option is intended to facilitate interoperating with Exchange itself, in particular, so that MTA-generated journal messages can be imported into Exchange.

Notify Actions in System-level Sieves No Longer Canceled

Notify actions in user-level sieves are automatically cancelled when the overall Sieve verdict is jettison, refuse, reject, or ereject. It used to be that notify in a system-level sieve would also be canceled, but no longer - such notify actions will now be honored regardless of the Sieve verdict, making it possible to use notify for some limited administrative auditing functions.

New Channel Options: noaddrtypescan, addrtypescan and addrtypescanbccdefault

If the `addrtypescan` channel option is set, `RCPT TO:` (envelope recipient) addresses are compared with header recipient fields (`To:`, `Cc:`, `Bcc:`, `Resent-To:`, `Resent-Cc:`, and `Resent-Bcc:`).

When a match is found, that fact is recorded in the delivery flags associated with that envelope recipient. The flags are then used when generating the report part of Exchange 2007 envelope journaling archive messages, which distinguishes between various types of envelope recipient addresses.

Because delivery flags are used to store this information, the MTA's delivery flag transfer facilities may be used to transport this information between MTAs.

The option, `addrtypescanbccdefault` operates in the same way as `addrtypescan`, except that when no matches are found for a given address, the address is assumed to be a blind carbon recipient. This option should only be used when it is certain the messages have come directly from a client that implements `Bcc:` by omitting the blind carbon recipient from the header and which doesn't support any form of local mailing lists. Use in any other context is guaranteed to result in incorrect types being attached.

`noaddresstypescan` is the default.

New Data File: tlds.txt

A new data file, `tlds.txt`, has been added to the information loaded by `imsimta chbuild`.

The file is a copy of the official IANA list of top level domains, available at: http://data.iana.org/TLD/tlds-alpha-by-domain.txt. The file is used for TLD validity checks.

New Rewrite Rule Metacharacters $, and $>

New rewrite rule metacharacters $, and $> have been added.

These metacharacters act like $V and $Z, respectively, except that instead of checking for a domain known to the directory, they check the top-level part of the current domain against the list of known valid TLDs. $, succeeds if the top-level part is on the list; $> succeeds if the top-level part isn't on the list.

internet.rules File Modified

The internet.rules file has been modified. All of the individual rules for top-level domains have been removed and replaced by the single rule:

```
. $U%$H$,$H@TCP-DAEMON
```

With this change it is no longer necessary to update `internet.rules` when additional TLDs are defined. Instead, obtain a new copy of `tlds.txt` from IANA, place it in the `config` directory, and run `imsimta chbuild`.

Handling and Default Values for Some Error Text Options Changed

The handling and default values for some error text options have been changed. The affected options and their new defaults are:

`error_text_block_over` "channel limit of %d kilobytes on message size exceeded"
`error_text_line_over` "channel limit of %d lines on message length exceeded"
`error_text_list_block_over` "list limit of %d kilobytes on message size exceeded"
`error_text_list_line_over` "list limit of %d lines on message length exceeded"
`error_text_user_block_over` "user limit of %d kilobytes on message size exceeded"
`error_text_user_line_over` "user limit of %d lines on message length exceeded"

> **Note**
> When the errors are triggered, the actual limit value in blocks or lines is substituted for the %d tags. The specific default depends on the value of the `block_size` MTA option. "kilobytes" is used if `block_size` is 1024 (the default), "megabytes" (1024*1024), "blocks" (>1) or "bytes" (1) are used as appropriate if the `block_size` value is changed.

Some new error text options have also been added:

`error_text_message_too_large` "a message size of %d kilobytes exceeds the size limit of %d kilobytes computed for this transaction"
`error_text_insufficient_disk` "message exceeds disk space available at this time"
`error_text_message_too_long` "a message %d lines long exceeds the line limit of %d lines computed for this transaction"

> **Note**
>
> Two %d substitutions are used. The first is the actual number of blocks/lines found in the message, while the second is the computed limit. Kilobytes will be replaced by other words depending on the `block_size` setting, as described above.

System-level Sieve Loop Constructs

A general loop construct is now available for use in system-level sieves. The basic syntax is:

```
loop {
...
exitif (<em>expression</em>);
...
}
```

A loop may contain zero or more `exitif` statements. Loops may be nested:

```
loop {
...
loop {
...
exitif (<em>expression1</em>); # Exit from inner loop #1
}
...
exitif (<em>expression2</em>); # Exit from outer loop
...
loop {
...
exitif (<em>expression3</em>); # Exit from inner loop #2
}
}
```

The loop construct is allowed only in system-level sieves, specifically spamfilter plugin, alias file [filter] nonpositional parameter, destination channel, *_ACCESS mapping, source channel, and system sieves.

Exercise extreme care when using loops. It is possible to put the MTA into a CPU loop with an incorrectly constructed loop.

New Channel Options: sourceconversiontag and destinationconversiontag

The new channel options `sourceconversiontag` and `destinationconversiontag` each accept a single argument - a comma-separated list of conversion tags. The source conversion tag list is attached to all message recipients. Any destination conversion tags for a given channel are attached to all recipients associated with that channel.

New Destination Channel Options: parameterformatdefault, parameterformatminimizeencoded, and parameterformatstripencoded

The channel option `parameterformatdefault` is the default and produces the default handling of RFC 2231 encoded words in content-type and content-disposition parameters. This results in the application of heuristics to determine when encoded words are used.

When `parameterformatminimizeencoded` is set, the use of encoded words is minimized. Encoded words not involving charset or language information or 8bit characters are replaced with regular parameter values.

The channel option `parameterformatstripencoded` eliminates the use of encoded or segmented parameters entirely.

New spamfilter MTA Options: spamfilter*N*_includeheaders (*N* = 1,8)

The allowed values for `spamfilter`*N*`_includeheaders (`*N*` = 1,8)` are `0` and `1`. A setting of `1` causes any headers added to the message by `$A` in the various `*_ACCESS` mapping to be passed to the spam filter as part of the regular message header. A setting of `0`, the default, disables this capability.

New Channel Option: msexchange

If set, `msexchange` enables various fix-ups and workarounds for various Microsoft Exchange compatibility issues. The MTA has been modified so the setting of this keyword on a destination channel instructs any MIME conversion operation to eliminate all `Content-disposition:` fields from

text/calendar message parts, because these fields interfere with properly handling of these attachments by Outlook.

Nonstandard :aindex Tag With Sieve Address Test

The nonstandard `:aindex` tag may now be specified with the Sieve address test. This tag accepts an positive integer as an argument. If a value <n> is specified, only the `<n>th` address in each header field will be tested.

> **Note**
> `:aindex` can be combined with `:index` to test a single address in a single header.

authrewrite and AUTH_REWRITE Enhancements

Enhancements have been made to the `authrewrite` channel option and `AUTH_REWRITE` mapping.

If bit 9 (value 512) of `authrewrite` is set, the final tag set by the `*_ACCESS` mappings will be prefixed to the `AUTH_REWRITE` mapping probe.

New MTA Options: scan_channel, scan_origin, and scan_recipient

These options control how the MTA is initialized when it is used for antispam and antivirus checks by other applications and utilities like `imexpire`.
`scan_channel` specifies the default source channel for such scanning operations. The l channel is used if no default is specified.
`scan_originator` specifies the MAIL FROM address used. (There is no envelope in this situation but the MTA requires one so a dummy one is constructed.) The default is the empty string.
`scan_recipient` specifies the RCPT TO address used. (A dummy envelope is constructed.) The default value for this option is `postmaster`.

New Sieve Environment Item: vnd.oracle.last-verdict

There is a new Sieve environment item, `vnd.oracle.last-verdict`.

When the sieves associated with a recipient are evaluated in order, each evaluation that performs an explicit handling action sets this item as it finishes so the next sieve in the sequence can check it. A script that doesn't perform an explicit handling action will leave this item unchanged.
Values that can be set are:

```
refuse
reject
ereject
jettison
fileinto
redirect
keep
discard
```

> **Note**
>
> Testing this environment item makes the script recipient-specific in the same way that an envelope `to` test does, and will result in this and subsequent sieves being re-evaluated for every recipient. Although any script can test this item, it is intended for use when the MTA is being used to perform antispam and antivirus checks by other applications and utilities like `imexpire`.

Scan Mode Added to the imsimta test mime Utility

A `scan` mode has been added to the `imsimta test -mime` utility.

This mode engages the MTA to perform a message scan in the same way `imexpire` does. The command syntax is:

```
imsimta test -mime -scan="<sieveexpression>" [-channel=<channelname>]
[input-message]
```

The channel defaults to setting of the `SCAN_CHANNEL` MTA option if `-channel` is not specified. Note, however, that AS/AV scanning only engages if `-channel` is explicitly specified.

New Tailor Variable: IMTA_HOST.

`IMTA_HOST` is set during MTA initialization to the value of the `local.hostname` `configutil` option. This tailor variable, like all other tailor variables, can be substituted into configuration files using the `&/variable-name/` construct, in this case `&/IMTA_HOST/`, eliminating the need to specify the host name everywhere and making configuration files more portable. The primary use of this new feature will be in generating initial XML configurations.

New Sieve Address Test Part Modifiers: :display and :comment

The Sieve address test now supports two new part modifiers: :display and :comment.

The `:display` modifier causes the test to operate on the display-name phrase that appears in front of an address enclosed in `<>s`.

The `:comment` modifier causes the test to operate on any parenthetical comments that appear after an address.

:raw and :text Modifiers Enhanced

The `:raw` and `:text` modifiers defined for the body test may now be used in header and address tests. As with body, `:raw` specifies that no MIME decoding be perform, while `:text` specifies that MIME decoding should be performed.

In the case of header and address, the MIME processing is the decoding of encoded-words. :text is the default for header and address tests on :comment or `:display` address parts. `:raw` is the default for other sorts of address tests.

MTA Address Reversal Logic Redesigned to Improve Subaddress Handling

The MTA's address reversal logic has been extensively redesigned to improve subaddress handling.

Previously, the presence of a subaddress would prevent address reversal from occurring. Now the default behavior is to attempt to match the address with or without the subaddress. If there is a match, the subaddress is transferred to any rewritten address.

This behavior may be explicitly specified by setting the `subaddressrelaxed` option on the source channel. The `subaddresswild` option, if set, matches against subaddresses but disables transfer of the subaddress to the rewritten address. Finally, `subaddressstrict` disables special subaddress handling during the reversal process.

MTA Folds Multiple To:, Cc:, and Bcc: Fields into a Single Field

Multiple recipient fields aren't allowed in generated message headers per RFC 5322, so the MTA has code to fold multiple `To:`, `Cc:`, and `Bcc:` fields into a single field. This logic has now been extended to apply to multiple `From:` fields, but only if a `Sender:` field is also present.

`From:` allows multiple values when a `Sender:` is also present, so a reasonable interpretation of multiple From: fields is that they specify multiple from values. This is not a reasonable inference if `Sender:` isn't present, because the result multivalued field will be illegal.

Support for RFC 5435, RFC 5436

Support has been implemented for RFC 5435 (Sieve Email Filtering: Extension for Notifications) and RFC 5436 (Sieve Notification Mechanism: mailto).

This change extends existing support for the notify action defined in draft-martin-sieve-notify-01.txt Both forms of notify can be used simultaneously, such as, scripts like:

```
require ["notify", "enotify"];
notify :message "subject" "mailto:a@b?body=body";
notify :method "email" :options "a@b" "subject" "body";
```

are allowed. If both notify extensions are enabled, the action arguments are examined to determine which extensions is being used.

Renamed job_controller Options

Some `job_controller` options have been renamed. `max_life_conns` is renamed to **`max_life_askwork`** and `max_life_age` is renamed to **`max_life_time`**.

Legacy configuration supports both names with the new name taking precedence, but XML configuration only uses the new names. The XML configuration migration tool `configtoxml` will automatically convert to the new names.

Some LDAP URL and DNs Errors Ignored

Errors in LDAP URLs or DNs encountered during group access check or `EXPN` command processing are now ignored

Previously, such errors halted the expansion process.

imsimta test rewrite Supports size=n Qualifier

`imsimta test -rewrite` now supports a `-size=n` qualifier than can be used to set message the

size as if the SMTP SIZE extension had been used.

This is useful for checking size-based restrictions.

Dispatcher Options Default Value Changes

For many dispatcher options, the product generated settings in initial configuration that did not match internal defaults. This aligned the dispatcher default settings to the initial configuration settings, eliminating the need for most initial configuration.

You will not notice a change in behavior unless you have removed initial configuration settings from `dispatcher.cnf`. Changes are as follows:

Option	Previous Default Value	Initial Configuration Value	New Default Value	New iconf
historical_time	3600	0	0	-
max_conns	10	50	50	-
max_idle_time	0	600	600	-
max_life_conns	300	10000	10000	-
max_life_time	86400	84000	86400	-
max_procs	5	10	10	10
max_shutdown	2	2	5	-
min_conns	3	30	30	-
max_conns	10	50	50	-
min_procs	0	1	1	-
backlog	conns*procs(5)	-	conns*procs (128)	-

Notes: the `max_shutdown` default was changed to half of `max_procs`. The `backlog` minimum changed from 5 to 128 to improve denial-of-service resistance.

Messaging Server Initial Configuration Modified to Generate Correct INTERNAL_IP Mapping Table

The Messaging Server initial configuration (`configure`) has been modified to generate a correct `INTERNAL_IP` mapping table.

A new *`statefile`* variable has also been introduced. See the topic on Messaging Server initial configuration in *Unified Communications Suite Installation and Configuration Guide*.

New MTA Option: LOG_AUTH

New `LOG_AUTH` MTA option has been introduced.

This option has the same basic semantics as `LOG_USERNAME`, except that if set, it logs the value of the `SMTP AUTH` parameter on enqueue, assuming one was specified and retained. On dequeue the `AUTH`

parameter is logged only if it is passed on to the remote SMTP server. The new field appears immediately after the user name in the old style log format. An `au` attribute is used in the new XML format.

If bit 1 (value 2) is set in the option, the field appears in the `LOG_ACTION` mapping probe immediately after the user name. `LOG_AUTH` defaults to `0` unless `LOG_FORMAT` is set to `4` - in that case, it defaults to `1`.

iSchedule Channel Handles iMIP Messages

The iSchedule channel now handles iMIP messages. See Using the iSchedule Channel to Handle iMIP Messages.

New Mapping: AUTH_ACCESS

A new mapping, `AUTH_ACCESS`, is consulted just prior to initiating SMTP client connections.

The mapping probe is:
`channel|filename|queue-time|envelope-from|auth-param|username|domain`

Parameter	Description
`channel`	the channel the message is being dequeued from
`filename`	full file name of the message
`envelope-from`	contains the envelope-from information
`queue-time`	the approximate time in seconds that the message has been in the queue (-1 if this cannot be determined)
`auth-param`	the unencoded value of any MAIL FROM AUTH= parameter associated with
`username`	the authentication identity used to submit the message
`domain`	the DNS name of the server the message is going to be sent to

The return value can contain a number of flags plus a series of |-separate fields. Setting a flag causes the consumption of zero or more fields, processed in the following order:

1. `$U` enables SMTP client debugging for this transaction. No fields are consumed.
2. `$N` aborts this connection attempt. One field is used to specify the error text.
3. `$F` overrides the message's envelope from address. One field is used to specify that address.
4. `$A` overrides the AUTH= parameter. One field is used to specify the override value.
5. `$Q` overrides the credentials used for plain authentication. Three fields are consumed by this flag: The first specifies an authorization identity (normally left blank), the second specifies an authentication identity (the `username`), and the third specifies a password.
6. `$/` changes the behavior in the event PLAIN authentication is attempted and fails. Normally this condition is ignored if `maysaslclient` is in effect and causes the message to bounce if `mustsaslclient` is in effect. Setting `$/` results in a temporary delivery failure in this case.
7. `$D` overrides the DNS name of the server the message will be sent to. One field is used to specify that DNS name
8. `$P` overrides the value given by the `port` channel keyword. (The default is port 25.) One field is used to specify the port number.
9. `$S` enables the use of smtps:. No fields are consumed. (The `SSL_CLIENT` channel option is ignored if either $D or $P is specified.)
10. `$X` disables MX lookups for connection establishment. No fields are used
11. `$M` enables random MX lookups for connection establishment. No fields are used.

12. $B overrides the value given by the lastresort channel keyword. One field is used to specify the last resort server name.
13. $T forces musttlsclient on for this message. No fields are used.
14. $H disables both maytlsclient and musttlsclient for this message. No fields are used.
15. $G forces mustsaslclient on for this message. No fields are used.
16. $I disables both maysaslclient and mustsaslclient for this message. No fields are used.
17. $Y is a no-op; it is useful for specifying mapping results that cause SMTP client processing to proceed normally.

New Delivery Option Modifier: %

A new delivery option modifier % has been introduced. It is the complement of the existing ^ modifier - ^ and causes the option to be active only between the specified start and end dates, whereas % causes the option to be active only outside those dates. In either case, the option remains active if no date restrictions are in effect.

MAX_NOTIFYS and MAX_VACATIONS MTA Options Modified

The definitions of the MAX_NOTIFYS and MAX_VACATIONS MTA options have been modified.

Previously these options were checked only at sieve evaluation time, so a sieve that used these actions would fail only at that point, and ihave tests (to see if the extensions are available) did not work.

A check is now made when processing require and ihave clauses and will fail if the option is set to zero.

Mapping Template Quoting Restrictions Removed

You are no longer required to use dollar signs to quote spaces and tabs in mapping templates.

New Delivery Option: nomail

A new option nomail has been added to the default set of delivery options.

This option, defined as:

```
#*&nomail=$M+$D@bitbucket
```

acts as a valid recipient but silently deletes all messages. This setting is useful for setting up LDAP entries with valid but unmonitored email addresses.

LMTP Server Supports STARTTLS.

The LMTP server now supports STARTTLS. This is controlled by the maytlssserver and musttlsserver channel keywords.

Out-encoding Clauses Now Honored While Relabeling

Out-encoding clauses are now honored while relabeling. When relabel is set to 1, out-encoding is no longer a no-op.

New In-encoding Clause Allowed in Conversion Files

A new in-encoding clause is now allowed in conversion files.

This clause can be used to restrict conversion entries to operate only on parts with the specified encoding.

SMS Gateway Server RFC 3458 Support

The SMS Gateway Server now supports RFC 3458, Message Context for Internet Mail.

A Message-context: header line is added to each SMS message which is gatewayed to email. The message context class is `pager-message` which both conveys that the e-mail message contains the text of an SMS message and is possibly urgent in nature.

MTA's Sieve External List Support Modified

The MTA's Sieve external list support has been modified to bring it into compliance with the latest draft.

1. The `SIEVE_EXTLISTS` mapping should set `$N` if the list name is recognized but the specified item cannot be found on the list. Failure to set `$N` in this case will result in a run-time error.
2. `$F` may used to force a run-time error to be reported. If this is done, the mapping return string should be set to the desired error text.
3. List names beginning with ":" automatically have the string `"urn:ietf:params:sieve"` `prepended`; this eliminates the need to duplicate tests for the two possible forms for such URLs the extension allows.
4. `$N` is assumed if `$Y` or `$F` isn't set for the mandatory-to-implement `urn:ietf:params:sieve:addrbook:default` and its abbreviated form `:addrbook:default`.
5. The `valid_ext_list` test has been implemented. The test results in a `SIEVE_EXTLISTS` mapping probe of the form: `valid_ext_list|owner|spare_4|spare_5|spare_6|listname`. The `$T` flag will also be set and can be used as another means of checking the type of probe.
6. The $S flag will be set if the list is being checked by a system-level sieve.

See the topic on Messaging Server support for externally stored lists Sieve extension in Features Introduced in Messaging Server 7 Update 1.

The addressbook example in item #273 has been updated to match this revision and to support valid_ext_list tests.

importancetest Sieve Test and importanceadjust Sieve Action Implemented

The `importancetest` Sieve test and `importanceadjust` Sieve action have been implemented.

These nonstandard extensions are provided so that multiple Sieve scripts can cooperate in making a determination of a message's importance, similar to how the `spamtest` and `spamadjust` extensions allow multiple Sieves to cooperate in determining whether or not a message is spam.

`importancetest` and `importanceadjust` work in the same way `spamtest` and `spamadjust` with these exceptions:

- Importance values range from 0 to 100.
- The initial value of `importancetest`, and the value if no `importanceadjust` actions have been performed, is 50.
- Fractional adjustments are allowed, but the importance value is rounded to an integer by the `importancetest` test.

imsimta test rewrite Utility Supports identifier Switch

The `imsimta test -rewrite` utility now supports a `-identifier` switch to allow testing of the handling of message identifiers as well as addresses. The prompt changes from `Address:` to {Identifier:}} when this switch is specified and values entered are processed according to the syntax rules and heuristics appropriate for message identifiers.

New MTA Option: LDAP_CHECK_HEADER

A new MTA option, `LDAP_CHECK_HEADER` has been added.

It specifies the name of a group attribute which is used to determine if the message header should be checked for duplication of list recipient addresses. The option does not have a default value.

If the specified attribute has a value of `jettison`, the list copy for the recipient specified in the header is jettisoned. If the value is `discard`, the system behaves as if the system Sieve had performed a discard on the list copy for that recipient.

The same capability is also available to aliases defined in the alias file or database via the new `header_check` nonpositional parameter.

This capability depends on address typing being enabled - either the `addrtypescan` or `addrtypescanbccdefault` channel option must be set on the source channel.

Although this capability has the potential to reduce unwanted message duplicates, **extreme** care should be exercised when it is used. Headers fields are trivially forged, making it possible to send a message that appears to have been sent to a certain recipient when in fact it has not. This could potentially be used to suppress the list copy for a given recipient.

New imsimta calc mm and imsimta test expression mm switch: system

The `imsimta calc -mm` and `imsimta test -expression -mm` commands now support a new `-system` switch.

If specified, the Sieve is treated as a system-level Sieve. If not, it is treated as a user-level Sieve.

Spaces Now Valid as Delimiters in Conversion Tag Strings

In addition to commas, spaces are now accepted as delimiters in conversion tag strings.

SMS Gateway Server Can Deliver Content in Subject Header Line

Gateway profiles may now request that the SMS Gateway Server place the entire content of an SMS message gatewayed to email only in the email message's Subject: header line. No content is placed in the email message's body.

Set the new `TEXT_TO_SUBJECT` option to `1` to enable this behavior.

When `TEXT_TO_SUBJECT=1`, the `EMAIL_BODY_CHARSET` option is ignored for that gateway profile. The default value of the `PARSE_RE_0` option becomes `[\t]([^(])[ \t](.)`.

New MTA Options: LOG_CONVERSION_TAG, LOG_IMAP_FLAGS, and LOG_DELIVERY_FLAGS

These are all bit-encoded options with the same basic semantics as the various other `LOG_*` options.

Setting bits in `LOG_CONVERSION_TAG` causes applicable conversion tags to be logged on enqueues and dequeues. This field appears after queue times (`LOG_QUEUE_TIME` option) and before IMAP flags (

LOG_IMAP_FLAGS option). The tg attribute is used in XML format logs. This field appears after conversion tags (LOG_CONVERSION_TAG option).

Setting bits in LOG_IMAP_FLAGS causes the IMAP flags that are to be set by the Sieve imap4flags extension to be logged. The associated XML attribute is if.

Setting bits in LOG_DELIVERY_FLAGS causes the various delivery flag bits to be logged as an integer. This field appears immediately after IMAP flags (LOG_IMAP_FLAGS option). The associated XML attribute is df.

SMTP Server Rejects STARTTLS When Followed by Additional Commands

The SMTP server rejects STARTTLS if there are additional commands piggybacked on it in the stream. The purpose of this is to avoid certain attacks. This use of STARTTLS is now counted as a bad command, which in conjunction with disconnection limits, can be used to force the connection to close if an attack is attempted.

New SMTP Channel Option: CLIENT_CERT_NICKNAME

This SMTP client option specifies the certificate to use for SMTP client authentication via STARTTLS. The nickname follows the same format as other certificate nicknames in Messaging Server. It is split at the first ":" character, and the portion prior to the colon is the security token; the portion subsequent to the ":" character is the certificate nickname in that security token. If no ":" character is present, then the NSS built-in certificate databases are used. Note that CLIENT_CERT_NICKNAME should only be used with IGNORE_BAD_CERT=0 set, as otherwise the security it can provide is defeated.

New Bits/Values Added to MTA Option: LDAP_USE_ASYNC

Bit	Value	Usage
16	(value 65536)	LDAP_AUTH_MAPPING* generated outer URLs (New in Messaging Server 7 Update 5)
17	(value 131072)	LDAP_AUTH_MAPPING* generated inner URLs (New in Messaging Server 7 Update 5)

New Bit/Value Added to MTA Options: USE_CANONICAL_RETURN and USE_ORIG_RETURN

Bit	Value	Usage
24	16777216	When set, use the original envelope From: address in mailing list LDAP_AUTH_MAPPING* mapping checks (New in Messaging Server 7 Update 5)

Setting the LOG_FORMAT MTA Option to 4 Now Causes Log Entries to Be Written in an XML-Compatible Format

A log entry appears as a single XML element containing multiple attributes and no sub-elements. Three elements are currently defined, en for enqueue/dequeue entries, co for connection entries, and he for header entries.

The following two-column table shows the attributes and descriptions enqueue/dequeue (en) elements can have:

Attribute	Description
ts	time stamp (always present)
no	node name (present if LOG_NODE=1)
pi	process id (present if LOG_PROCESS=1)
sc	source channel (always present)
dc	destination channel (always present)
ac	action (always present)
sz	size (always present)
so	source address (always present)
od	original destination address (always present)
de	destination address (always present)
rf	recipient flags (present if LOG_NOTARY=1)
fi	filename (present if LOG_FILENAME=1)
ei	envelope id (present if LOG_ENVELOPE_ID=1)
mi	message id (present if LOG_MESSAGE_ID=1)
us	username (present if LOG_USERNAME=1)
au	AUTH parameter (present if LOG_AUTH=1)
ss	source system (present if LOG_CONNECTION = 0 and source system information is available)
se	sensitivity (present if LOG_SENSITIVITY=1)
pr	priority (present if LOG_PRIORITY=1)
in	intermediate address (present if LOG_INTERMEDIATE=1)
ia	initial address (present if LOG_INTERMEDIATE=1 is set and intermediate address information is available)
fl	filter (present if LOG_FILTER=1 and filter information is available)
re	reason (present if LOG_REASON=1 and reason string is set)
di	diagnostic (present if diagnostic info available)
tr	transport information (present if LOG_CONNECTION=5 is set and transport information is available)
ap	application information (present if LOG_CONNECTION=6 is set and application information is available)

Here is a sample en entry:

```
<en ts="2004-12-08T00:40:26.70" pi="0d3730.10.43" sc="tcp_local"
dc="l" ac="E" sz="12" so="info-E8944AE8D033CB92C2241E@example.com"
od="rfc822;ned+2Bcharsets@example.com"
de="ned+charsets@example.com" rf="22"
fi="/path/ZZ01LI4XPX0DTM00IKA8.00" ei="01LI4XPQR2EU00IKA8@example.com"
mi="&lt;11a3b401c4dd01$7c1c1ee0$1906fad0@elara&gt;" us=""
ss="example.com ([208.250.6.25])"
in="ned+charsets@example.com" ia="ietf-charsets@example.com"
fl="spamfilter1:rvLiXh158xWdQKa9iJ0d7Q==, addheader, keep"/>
```

Note that this entry has been wrapped for clarity; actual log file entries always appear on a single line.

The following two-column table shows the attributes and descriptions connection (co) entries can have:

Attribute	Description
ts	time stamp (always present, also used in en entries)
no	node name (present if LOG_NODE=1, also used in en entries)
pi	process ID (present if LOG_PROCESS=1, also used in en entries)
sc	source channel (always present, also used in en entries)
dr	direction (always present)
ac	action (always present, also used in en entries)
tr	transport information (always present, also used in en entries)
ap	application information (always present, also used in en entries)
mi	message ID (present only if message ID information available, also used in en entries)
us	user name (present only if user name information available, also used in en entries)
di	diagnostic (present only if diagnostic information available, also used in en entries)

Here is a sample co entry:

```
<co ts="2004-12-08T00:38:28.41" pi="1074b3.61.281" sc="tcp_local" dr="+"
ac="O" tr="TCP|1.1.1.1|25|1.1.1.1|33469" ap="SMTP"/>
```

The following two-column table shows the attributes and descriptions header (he) entries can have:

Attribute	Description
ts	time stamp (always present, also used in en entries)
no	node name (present if LOG_NODE=1, also used in en entries)
pi	process id (present if LOG_PROCESS=1, also used in en entries) |
va	header line value (always present)

Here is a sample he entry:

```
<he ts="2004-12-08T00:38:31.41" pi="1074b3.61.281" va="Subject: foo"/>
```

Support for the Vacation Seconds Extension

As part of implementing this extension, described in RFC 6131, two new MTA options, VACATION_MINIMUM_TIMEOUT and VACATION_MAXIMUM_TIMEOUT, have been added. The VACATION_MINIMUM_TIMEOUT option establishes a minimum value, in seconds, for the vacation :days and :seconds parameters. Values lower than the minimum are silently adjusted up to the minimum; no error occurs. The default value for VACATION_MINIMUM_TIMEOUT is 0.

The VACATION_MAXIMUM_TIMEOUT option sets a maximum value for vacation timeouts. Values larger than the maximum are silently adjusted down to the maximum; no error occurs. The default value for VACATION_MAXIMUM_TIMEOUT is 2^31-1.

Support Added for a Domain-Level Alias detour host optin Attribute

This attribute has semantics similar to the user-level detour host optin attribute selected by the LDAP_DETOURHOST_OPTIN MTA option, except that it appears on the recipient's domain entry rather than the recipient's user entry. A new MTA option LDAP_DOMAIN_ATTR_DETOURHOSTOPTIN has been added to specify the name of this domain-level attribute. The default value for this MTA option is the empty string, meaning no such attribute is defined by default.

Any user-level detour host optin will be honored before domain-level optins are checked. The channel level aliasdetourhost keyword is only honored if no user or domain level optins occur.

Source Channel Based Alias Detour Processing Now Works on Alias File and Alias Database Expansions

New Bit/Value Added for MTA Option: INCLUDE_CONVERSIONTAG

Bit 8 (Value 256) of the INCLUDE_CONVERSIONTAG MTA option, if set, causes the current set of conversion tags to be included in REVERSE mapping probes. The tags are comma separated and delimited by vertical bars. They appear after any source or destination channel information but before the actual address in the probe, e.g.,

```
src-chan|dest-chan|tag1,tag2,tag3...|address
```

Conversion tag information only appears in probes that occur as messages are written to a queue. Other rewriting operations do not include the tags in the probe.

In order to be able to conveniently test such mappings, a -tag=value qualifier has been added to imsimta test -rewrite. This qualifier will set the conversion tag string that is used in REVERSE mapping probes (assuming, of course, that bit 8 of INCLUDE_CONVERSIONTAG is set).

New TCP Channel Option: SSL_CLIENT

If set to 1, SSL/TLS negotiation is performed immediately after a connection is established by the SMTP client. The default is 0, meaning this will not be done.

SSL_CLIENT is useful for establishing point-to-point links to other systems using smtps: on port 465.

Facilities Provided for Selecting Different Capture Formats for Capture Requests

Made in Mappings or Through LDAP

- New MTA option CAPTURE_FORMAT_DEFAULT. This option specifies the default capture format for capture actions implemented via an LDAP_CAPTURE attribute. The default value of 0 causes the regular report format to be used, while a value of 1 switches the default to Exchange journal format.
- LDAP attribute tags can be used on any LDAP_CAPTURE attributes to explicitly specify the capture format to use. An attribute tag of ;format-report explicitly calls for the regular report format, while an attribute tag of ;format-journal explicitly requests Exchange journaling format.
- The $J metacharacter in SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS mappings switches any capture request made by that particular mapping from report to Exchange journal format.
- The $+L metacharacter in FROM_ACCESS mappings switches any capture request made in that mapping from report to Exchange journal format.

RESETDEBUG Option Added to Archiving Spamfilter Plugin

The RESETDEBUG option available in the Milter spamfilter plugin has been added to the archiving spamfilter plugin.

Bit/Value of MTA Option: LOG_MESSAGES_SYSLOG Sets syslog Priority and Facility Code

Only the low 16 bits of the absolute value of the log_messages_syslog option are now used to set the syslog priority and facility code. Bit 16 (value 65536), if set, disables sending header logging to syslog.

IP_ACCESS Mapping Includes Channel the Message Is Being Dequeued From

The IP_ACCESS mapping now includes the channel the message is being dequeued from. Previously the mapping included the name of the channel that enqueued the message.
Bit 1 (value 2) of the USE_IP_ACCESS MTA option, if set, causes the the name of the channel that enqueued the message to be included in the probe.

MMP configutil support And Configuration Simplification

The MMP honors most authentication and LDAP-related configutil settings (such as sasl.*, local.ugldaphost, local.ugldapbinddn, and so on) if the equivalent setting is not also set in the separate MMP configuration file. As a result, there is no longer a need for the files ImapProxyAService-def.cfg, PopProxyAService-def.cfg, and SmtpProxyAService-def.cfg in the configuration directory. Consequently, the generated initial configuration is simpler and less redundant.

The POP and IMAP proxies can operate without separate configuration files as long as AService.cfg and msg.conf have appropriate settings.

The SMTP Submission proxy still requires non-empty default:SmtpProxyPassword and default:SmtpRelays settings before it can operate.

In general, these changes are backwards compatible, except on a system where configutil settings have been explicitly set which impact authentication while the equivalent MMP settings are not set.

The following MMP settings are deprecated in favor of the relevant configutil settings. For backwards compatibility, the MMP setting is used instead of the configutil setting if present, but use of these MMP settings in new deployments is discouraged in favor of configutil settings. See the following table:

Deprecated MMP setting	Preferred Replacement Setting(s)
default:BindDN	local.ugldapbinddn
default:BindPass	local.ugldapbindcred
default:LDAPTimeout	local.ldapsearchtimeout
	local.ldapmodifytimeout
default:LdapUrl	local.ugldaphost
	local.ugldapusessl
	local.ugldapport
	service.dcroot
default:UserGroupDN	local.ugldapbasedn
default:SSLCacheDir	local.ssldbpath
default:SSLCertPrefix	local.ssldbprefix
default:SSLKeyPasswordFile	local.ssldbpath
default:SSLKeyPrefix	local.ssldbprefix

The following MMP settings now have equivalent `configutil` settings. The MMP setting takes precedence over the `configutil` setting for the MMP proxy service:

MMP Setting	Equivalent `configutil` Setting(s)
default:AuthCacheTTL	`service.authcachettl`
default:AuthenticationLdapAttributes	`sasl.default.authenticationldapattributes`
default:AuthenticationServer	`sasl.default.authenticationserver`
default:CRAMs	`sasl.default.ldap.has_plain_passwords`
default:DebugKeys	`local.debugkeys`
default:DefaultDomain	`service.defaultdomain`
default:DomainSearchFormat	`sasl.default.ldap.searchfilter`
default:HostedDomains	`sasl.default.ldap.searchfordomain`
default:LdapCacheSize	`service.authcachesize`
default:ReplayFormat	`local.*.replayformat`
default:RestrictPlainPasswords	`service.*.plaintextmincipher`
default:SSLAdjustCipherSuites	`local.ssladjustciphersuites`
default:SSLCertNicknames	`encryption.rsa.nssslpersonalityssl`
default:StoreAdmin	`store.admins`
default:TCPAccess	`service.*.domainallowed`
	`service.*.domainnotallowed`
default:VirtualDomainDelim	`service.loginseparator`
default:BeTheUser	`local.serveruid` (or `IMTA_USER` from `imta_tailor`)

Note
Note that `.*.` options use `popproxy`, `imapproxy` or `submitproxy` in place of `*` for the POP, IMAP and SMTP submit proxies respectively.

Note
The MMP only supports one StoreAdmin, whereas `store.admins` may be a space-separated list. If `store.admins` is a list, then it is mandatory to set default:StoreAdmin to use MMP features that require a store admin (such as certificate authentication, CRAM-MD5, PLAIN proxyauth authentication). The MMP's default:StoreAdmin setting now defaults to `admin` when neither it nor `store.admins` is explicitly set.

The following MMP authentication features do not have equivalent `configutil` settings:

MMP Settings
default:CanonicalVirtualDomainDelim
default:PreAuth
default:SearchFormat
default:StoreAdminPass
default:TCPAccessAttr

The following `configutil` settings without an MMP equivalent now impact the MMP's behavior:

`configutil` Setting	Discussion
`local.mmp.enable` Available for several releases.	If set, start-msg will start MMP
`local.watcher.enable` Available for several releases.	If set, MMP will connect to watcher
`local.watcher.port` Available for several releases.	Change default watcher port
`service.listenaddr` Available for several releases.	Address of interface MMP uses to connect to watcher
`local.ldapcheckcert` Introduced in **Messaging Server 7 Update 4**.	Set to 0 to ignore LDAP server cert errors

`local.domainmap.debuglevel` Introduced in **Messaging Server 7 Update 5**.	How much domain map debugging to output when loglevel is set to "debug". MMP would set this to 2 previously. Values up to 5 presently work and produce additional output
`local.ldapconnecttimeout` Introduced in **Messaging Server 7 Update 5**.	Set timeout for LDAP connect operations
`local.ldaptrace` Introduced in **Messaging Server 7 Update 5**.	Turns on LDAP transcripts (same as "ldap" debugkey)
`local.hostname` Introduced in **Messaging Server 7 Update 5**.	Override's MMP's opinion of local fully qualified host name
`sasl.default.auto_transition` Introduced in **Messaging Server 7 Update 5**.	Set user passwords in LDAP after successful authentication

MMP Honors configutil Settings

The following table describes the equivalent `configutil` settings for MMP settings:

MMP Setting	Equivalent `configutil` Setting
default:logdir	`logfile.*.logdir`
default:logfacility	`logfile.*.logfacility`
default:loglevel	`logfile.*.loglevel`
default:rollovertime	`logfile.*.rollovertime`
default:expirytime	`logfile.*.expirytime`
default:maxlogfilesize	`logfile.*.maxlogfilesize`
default:maxlogfiles	`logfile.*.maxlogfiles`
default:maxlogsize	`logfile.*.maxlogsize`
default:filemode	`logfile.*.filemode`

where * is `mmp` for the top-level AService log, `imapproxy` for the IMAP proxy log, `popproxy` for the POP proxy log and `submitproxy` for the SMTP submission proxy log. If both are set, the MMP setting takes precedence.

The `default:logdir` setting for the MMP accepts relative paths (relative to the `data` directory), while `logfile.*.logdir` **does not. Also note that `default:loglevel` accepts numeric values (for backwards compatibility) as well as the traditional string values, while `logfile.*.loglevel`** only accepts the traditional string log level values.

The IMAP, POP and SMTP Submission proxy nslog log files will inherit settings from the top-level MMP log file. Proxy-specific settings take precedence over MMP-wide settings.

> **Note**
> If the deprecated legacy MMP log format is used (default:usenslog 0), then `configutil` settings are ignored and only the `logdir` and `loglevel` settings impact the proxy.

Support for IMAP STARTTLS When Proxies (MMP, mshttpd, IMAP shared folders) Connect to Back-end

When `service.imap.plaintextmincipher` is set to a non-zero value, the back-end IMAP server does not permit password authentication without use of SSL or STARTTLS. The MMP, mshttpd and IMAP shared folder functions detect this situation and attempt to negotiate SSL/TLS prior to authenticating. SSL must still be enabled for the relevant proxy (although the client is not required to negotiate SSL for this to work, nor does the proxy have to be configured to require SSL for authentication). For example, `ImapProxyAService.cfg` should contain `default:SSLEnable yes`.

> **Note**
> POP and SMTP submission proxies do not have this feature.

MMP Support for IMAP Response Codes (RFC 5530)

MMP support for IMAP response codes (RFC 5530) has been implemented.

See the following examples of two IMAP response codes:
A001 login user1@domain1.test passwood

A001 NO [AUTHENTICATIONFAILED] Authentication failed

A002 login user1@domain1.test password
A002 NO [UNAVAILABLE] Service temporarily unavailable

MMP and IMAP Performance Improved on Solaris by Use of Solaris Event Ports

On Solaris, event ports are now used by default instead of the `poll()` system call. This improves MMP and IMAP scalability.

MMP Scalability Improvements: mmp.numprocesses

A single MMP process is not as effective as multiple MMP processes in using the resources of a multicore processor. To create multiple MMP processes, set `mmp.numprocesses` to the number of processes you want. The default value of `mmp.numprocesses` is 1.

For installations using the legacy `configutil` configuration method instead of Unified Configuration, the only way to change the value of `mmp.numprocesses` is to use the `service.mmp.numprocesses configutil` setting.

Note that setting `mmp.numprocesses` to a value greater than 1 is incompatible with POP before SMTP and that POP before SMTP is now deprecated (see the topic on deprecated and removed features for Messaging Server in *Unified Communications Suite Installation Guide*.)

IMAP Authentication Messages Now Include Session ID

IMAP Authentication messages now include the session ID. You can optionally include the session ID in the IMAP response text using `imap.logauthsessionid` (for Unified Configuration) or `local.imap.logauthsessionid` (for the legacy configuration method).

Ability to Disable Remote POP Collection for Webmail Has Been Added

Setting the `configutil` parameter `local.service.http.allowcollect` to 0 returns an error to the client upon an attempt to use the `collect.mjs` command.

For example:

```
POST /collect.mjs HTTP/1.0
Cookie: webmailsid=MEs3kDLRqBM
Content-Type: application/x-www-form-urlencoded
Content-Length: 87

mbox=INBOX&host=127.0.0.1&port=110&user=user4%40domain1.com&password=password&d
200
Date: Wed, 09 Jun 2010 19:59:12 GMT
Content-type: text/javascript; charset=utf-8
Content-length: 70
Last-modified: Wed, 09 Jun 2010 19:59:12 GMT
Pragma: no-cache
Expires: 0
Cache-Control: no-cache

while(1);
[646976,'Remote mail collection disabled by administrator']
```

Support For Third-Party-Authentication Co-Process

Added options so the store (imap/pop) supports the third-party-authentication co-process the way the MMP does. New options are: `sasl.default.authenticationserver` and `sasl.default.authenticationldapattributes`. These options also work for SMTP. To debug third-party authentication with the store, set `local.debugkeys` to include `authserv` (or `authserv hula`) and set `logfile.imap.loglevel` and `logfile.pop.loglevel` to debug. Now the third-party authentication server co-process is supported throughout the product.

The libibiff and libjmqnotify Notification Plugins Are Now Built-In to libstore

As a result, `libstore` now links against `libens`. The `local.store.notifyplugin` option now ignores all path names and only looks at the `libname`.

Password Expiration and IMAP Expiration Warning ALERTs

When LDAP password policy is enabled, as specified by: `http://tools.ietf.org/html/draft-behera-ldap-password-policy-10`, all Messaging Server traditional password authentication that uses LDAP Bind operations requests password policy status from LDAP (this includes IMAP, POP, mshttpd, MMP, and SMTP AUTH; for other products, such as Convergence, see the product documentation). In the event that a user's password has expired, the appropriate error message is recorded in the log and sent over protocol (Note: Not all protocols have a standardized password-expired error status/token at this time).

To enable LDAP password policy (including password expiration), see the Directory Server documentation. Make sure that all administrative accounts fall under a separate password policy or are otherwise excluded from password expiration. In particular, the store administrators required for Convergence, IMAP shared folders, MMP client certificate authentication, and other administrative proxy identities, the indexer admininstrator(s) if using ISS, and the Messaging end-user administrator for each server (as specified in `local.ugldapbinddn`).

In addition, the IMAP server can be configured to generate an IMAP ALERT when a user's LDAP password is about to expire. The IMAP specification, RFC 3501, requires compliant clients to display IMAP ALERTs to the user. Once this feature is enabled, an IMAP ALERT is generated either once per day per IMAP process the customer encounters (default) or, if Metermaid is used, the ALERT appears once per Metermaid expiration period. To enable this feature, use:

```
configutil -o local.imap.pwexpirealert.firstwarn -v 604800
```

This enables daily warnings when the password will expire in less than seven days (604800 is 7*24*60*60) and the Directory Server has provided that information to the IMAP server. To track alerts using Metermaid, configure each IMAP server system as follows:

```
configutil -o local.imap.pwexpirealert.viametermaid -v 1 configutil -o
metermaid.config.serverhost -v hostname.example.com configutil -o
metermaid.config.secret -v password
```

where `hostname.example.com` is the name of the host with the Metermaid server and `password` is replaced with your Metermaid password. Then on the Metermaid server machine configure as follows:

```
configutil -o local.metermaid.enable -v 1
configutil -o metermaid.config.secret -v password
configutil -o metermaid.table.pwexpirealert.data_type -v string
configutil -o metermaid.table.pwexpirealert.quota -v 1
configutil -o metermaid.table.pwexpirealert.quota_time -v 86400
```

Note that the default Metermaid table name for IMAP password-expiration alerts is `pwexpirealert`. To use a different table name, the `local.imap.pwexpirealert.metermaidtable` option is set. To specify a URL at which the user's password can be changed to include in the IMAP ALERT message, set `gen.pwchangeurl`.

The following options are also honored for Metermaid password expiration IMAP ALERTs (see the `configutil` reference for details):

```
metermaid.mtaclient.serverport
metermaid.mtaclient.serverhost
metermaid.mtaclient.readwait
metermaid.mtaclient.connectwait
metermaid.mtaclient.connectfrequency
metermaid.mtaclient.maxconns
```

If problems arise with use of Metermaid by IMAP for this purpose, the following tokens may be included in the `local.debugkeys` list to enable additional debugging to the IMAP log:

- metermaid: log a Metermaid client protocol trace at debug level
- connect: additional TCP connection diagnostics

The following is an example of an IMAP ALERT (line breaks are for editorial clarity; the IMAP server sends the alert as one line):

* NO [ALERT] Your password will expire in 2 days. You may change your password by directing your web rowser to `http://changepass.example.com`.

Note that there is no standard for communicating pending IMAP/POP/SMTP password expiration to mail clients when using traditional password authentication. Such a standard could be developed, if mail-client vendors were interested in it.

New C <channel> Option

A new -c <channel> option has been added to imexpire to specify an mta channel which, in turn, is used to specify spam filters to activate. Any source-channel spam filters configured on the channel are applied to each message that is scanned and any `spamadjust`/`spamtest`, `virusset/{{virustest`, or added headers are then visible to the Sieve expression used to expire messages. For example, suppose the channel invokes `SpamAssassin`, which is then configured to perform a `spamadjust` to communicate its results. In this scenario, an expression of the form

```
require \["comparator-i;ascii-numeric", "relational", "spamtest"\];
spamtest :value "ge" :comparator "i;ascii-numeric" "5"
```

will expire any message that received a spam score of 5 or more.

imexpire Enhancements

The following two-column table shows the the added attributes and descriptions to imexpire to facilitate

spam/virus scanning through a MTA channel:

Attribute	Description
channel	MTA channel (this replaces the -C <channel> option)
rescanhours	rescan messages that have not been scanned for the specified number of hours

Added support for body scan (ER 12266635).

imexpire by IMAP User Flags

Add an attribute group `userflag` to specify user flags in expire rules. Valid values are `and` and `or`. The usage is the same as the `Seen` and `Deleted` flag attributes.

Syntax:

```
userflag.<flagname>: and|or
```

As an example, the following rule expires messages with the `junk` flag set and are older than 30 days.

```
messagedays: 30
userflag.junk: and
```

Mechanism to Schedule Cleanup Immediately

A mechanism for scheduling an immediate mailbox cleanup has been added to `mboxutil`:

```
mboxutil -C [-g num] [-E encoding] [-p <MUTF7 IMAP pattern> | -P <regular
expression>]
-g <num> minumum cleanup size, num must be > 0 (default is store.cleanupsize)
-E <encoding> mailbox name encoding
-p <pattern> check mailboxes that match the specified MUTF-7 IMAP pattern
-P <regexp> check mailboxes that match the specified regular expression
```

Normally, cleanup is scheduled automatically when mailboxes are expunged. This command can be used to schedule cleanup manually when:

- The storage is very full.
- `store.cleanupsize` is reduced.

For example, the following command schedules cleanup of mailboxes with at least 50 expunged messages. Messages are removed when `store.cleanupage` has expired.

```
$ mboxutil -C -g 50
```

ENS Events Include Quota Information

ENS Events triggered from modifications to the message store now include quota information.

Features Introduced in Messaging Server 8.0

Features Introduced in Oracle Communications Messaging Server Version 8.0

Messaging Server 8.0 includes the following new features:

- Support for the Vacation-Seconds Extension (RFC 6131)
- imsimta test -mime Includes Additional -flags Switch
- $V Metacharacter Added to the FORWARD Mapping
- New P Logging Action Added
- Sieve :regex Tests Now Set Variables in the Same Way :matches Tests Do
- Default Change for the ignoremultipartencoding Channel Option
- Sieve extlists Tests Return 1024 Octets of Property Values
- TLS Cipher Name Now Uses the Full Standard TLS Cipher Suite Name Instead of the Short Form of the Name
- $R Input Flag Set in *_ACCESS and AUTH_REWRITE Mappings for Internal Channels
- Support For the MT-PRIORITY SMTP Extension Defined in RFC 6710
- New clonehosts Channel Option
- Use of Attributes Can Now Be Conditional on the Presence or Absence of Specific Object Classes
- New $]]]LDAP-URL[LDAP Query URL Mapping Template Substitution
- New Flag Inputs to the AUTH_REWRITE Mapping
- Connection Retry After STARTTLS Negotiation Failure
- New receivedstate Channel Option
- New implicitsaslexternal and explicitsaslexternal Channel Options
- The i;ascii-integer, i;ascii-casemap-collapse, and i;octet-collapse Comparators Now Supported by the MTA's Sieve Implementation
- New passsyntaxerrors and fixsyntaxerrors Source Channel Options
- Support Added for the foreveryport Sieve Extension
- Additional Settings for the LOG_USERNAME MTA Option
- Support Added for the extracttext Sieve Extension
- Support Added to Sieve and Recipe Language for User-Defined Routines
- Support Added for the Postfix's XCLIENT SMTP Extension
- BINARYMIME SMTP Extension Supported for Message Submission
- Security Enhancements to the BURL_ACCESS Mapping Table
- Enhanced Functionality for the $}...{ Mapping Metacharacter
- Enhancements to the MESSAGE-SAVE-COPY Mapping
- New LOG_TIMES MTA Option
- Field-Specific Address Reversal and Personal Name Rewriting
- New INCLUDE_SPARES2 MTA OPTION ADDED. INCLUDE_SPARES Renamed to INCLUDE_SPARES1
- Bits 7 and 8 Added to USE_FORWARD_DATABASE MTA Option
- New headercut Destination Channel Option
- nonotify Sieve Action Added
- Change to $N in the IP_ACCESS Mapping
- Correction to IP_ACCESS Mapping
- $L in a URL Substitution Now Accepts an Integer Argument
- Change to the Default Value of the Hold Delivery Option
- Change to Logging of SPF HELO/EHLO Check Failures
- memcache Now Supported as a Place to Store Vacation Timeout Information
- New $?whatever? URL Substituion
- New SIEVE_RECEIVED MTA Option
- Additional Settings for the USE_REVERSE_DATABASE MTA Option
- Timers Added to the MTA
- Support Added for the Sieve Duplicate Extension
- Support for Accessing Sleepycat Databases has been Removed from the MTA.

- New ACCESS_AUTH MTA Option
- $+C Flag in the FROM_ACCESS Mapping
- New Logging for setconversiontag and addconversiontag Sieve Actions
- Maximum MTA Symbol Table Increased
- Nonstandard translactionlog Sieve Action Implemented
- Some TCP Channel Options Now Default to 1 Instead of 0
- New 64-bit Counters Added
- Nonstandard `setenvelopefrom` Sieve Action Implemented
- New LDAP_GROUP_LAST_ACCESS_TIME MTA Option
- Acceptance of Illegal Date: Header Fields
- Nonstandard XUNAUTHENTICATE Extension and XUNAUTHENTICATE Command Added
- draft-ietf-appsawg-rrvs-header-field-01 Implemented
- New DEFER_HEADER_ADDITION MTA Option
- Change to the mgrpAddHeader Attribute
- New CONNECT_TIMEOUT spamfilter Plugin Option
- New LOG_FUTURERELEASE MTA Option
- New Qualifiers for imsimta test -expression and imsimta calc
- Sieve Variable and Encoded Character Substitutions Can Now be Performed on Nonstandard Functions
- Support for draft-delany-nullmx-01.txt Implemented
- MeterMaid's Client check_metermaid.so Now Supports Multiple MeterMaid Servers and SSL for Communication
- New affinitylist Channel Option
- A get_path(r) Function Added to Recipe Language
- Message Tracking and Recall
- New Nonstandard Sieve Override
- A :header Argument Can Now be Used with capture :dsn and capture :journal
- Largest Token in Command Line Reader Increased
- New ALLOW_SESSION_BLOCKS and ALLOW_TRANSACTION_BLOCKS SMTP Channel Options
- New destinationpassthrough Channel Option
- $O Flag in AUTH_ACCESS Mapping
- New Functionality for Sieve reject, ereject, and refuse Actions
- New setoperation Sieve Action
- Additional Bit Defined in the spamfilterX_received Set of Options
- New Sieve Warning Facility
- Support for draft-ietf-appsawg-email-auth-codes-07 Implemented
- Milter Extension to Allow Per-Recipient Modification Actions
- UNAUTHENTICATE Command Disabled by Default.
- Messaging Server Supports IMAP LIST Extension For Special-Use Mailboxes
- Conversion Channel Enhancements
- MTA's Internal Buffers Increased
- nonlocal mailDomainStatus Setting
- New Sieve Environment Item vnd.oracle.tracking-id
- New Sieve Environment Item vnd.oracle.message-hash
- New LDAP_DOMAIN_ATTR_SUBADDRESS MTA Option Added
- New -source=source-channel and -rsecret=recall-secret Qualifiers
- New USE_TEMPORARY_ERROR MTA Option
- New Channel Counters for Authentication Successes and Failures
- Enhanced imsimta test -rewrite Output
- imquotacheck -p <partition> Option to Report Quota for the Specified Partition.
- imcheck -B <bound> Option Added
- mboxutil -B <bound> Option Added
- imsconnutil -k Option Added
- imsbackup Reports the Amount of Data Backed Up Per Mailbox in Verbose Mode
- imsbackup Gives the Progress Information About Backup
- jmqnotify Expects Acknowledgement by Default for Persistent Messages
- ensnotify Has diskquota and diskquota Parameters in Messages
- Text Message Preview Cache

- Message Store Transaction Logging
- Log Rollover Manager
- SSL Support Added to IMAP Search When Communicating with Index and Search Service
- Charset Name IMAP-MODIFIED-UTF-7 replaced by IMAP-mailbox-name
- Added checksum Field to the Mailbox Header
- ENS Generates a New Notification on Folder Rename Event
- Changes to IMAP APPEND
- NSS version check
- Change to the Default SSL/TLS Cipher Suites
- STARTTLS Option for All LDAP Connections
- SSLv3 Disabled By Default
- immonitor-access Has SSL and SASL Support
- Changes to restricted.cnf, the Pipe Channel, and Privileged Shared Libraries.
- Bundled NSS Upgraded to NSS 3.17.4
- Message Store Manual Failover

Support for the Vacation-Seconds Extension (RFC 6131)

Support for the vacation-seconds extension, described in RFC 6131, has been implemented. As part of implementing this extension, two new MTA options, `VACATION_MINIMUM_TIMEOUT` and `VACATION_MAXIMUM_TIMEOUT`, have been added. The `VACATION_MINIMUM_TIMEOUT` option establishes a minimum value, in seconds, for the vacation `:days` and `:seconds` parameters. Values lower than the minimum are silently adjusted up to the minimum; no error occurs. The default value for VACATION_MINIMUM_TIMEOUT is 0. All of these capabilities are also available in Messaging Server 7.0.5.28.

The `VACATION_MAXIMUM_TIMEOUT` option sets a maximum value for vacation timeouts. Values larger than the maximum are silently adjusted down to the maximum; no error occurs. The default value for `VACATION_MAXIMUM_TIMEOUT` is 2^31-1.

imsimta test -mime Includes Additional -flags Switch

The "scan" mode of `imsimta test -mime` has been enhanced with an additional `-flags` switch. This switch takes a single quoted argument specifying the default set of IMAP flags associated with the message. This can be used to try out the `hasflag` test in an `imexpire`-compatible way.

$V Metacharacter Added to the FORWARD Mapping

The FORWARD mapping has been extended to support the $V metacharater. If set this metacharacter prevents a successful "forward and continue alias expansion" (YD) from setting the `via_alias` flag and thus defeating any `viaaliasrequired` check.

New P Logging Action Added

A new "P" type of logging action has been added. This action records requests to generate delivery status notifications. (Note that in some cases no actual DSN will end up being sent.) P records are only generated in cases where the error causing the DSN isn't recorded in any other log entry. The various cases where this happens are further detailed by the presence of a modifier character on the action. Currently the defined modifiers are:

F - address errors detected during alias or mailing list expansion operations (this includes bad RCPT TO addresses that are allowed because the acceptalladdresses channel option is in effect)
X - capture operations
J - journal operations
Y - sieve syntax or evaluation errors
D - success delivery receipts

Sieve :regex Tests Now Set Variables in the Same Way :matches Tests Do

Note that unlike glob-style matches, where the default is to store whatever matched any wildcard that appears in the pattern, in regex tests only regular expressions enclosed in parentheses are stored. If parentheses are needed but storage is not desired, the (?:) form may be used.

Default Change for the ignoremultipartencoding Channel Option

The `ignoremultipartencoding` channel option is now the default.

Sieve extlists Tests Return 1024 Octets of Property Values

Sieve `extlists` tests can now return up to 1024 octets of property values. This is the combined length of all properties and delimiter characters. This was limited to 256 octets in prior releases.

TLS Cipher Name Now Uses the Full Standard TLS Cipher Suite Name Instead of the Short Form of the Name

The TLS cipher name included in the application information string will now use the full standard TLS cipher suite name, instead of a short form of the name. This makes logging more informative and provides additional information for mappings using this string. However, in the unlikely event customers have written mappings that depend on the abbreviated cipher name, they may need to be updated.

$R Input Flag Set in *_ACCESS and AUTH_REWRITE Mappings for Internal Channels

The $R input flag will be set in *_ACCESS and AUTH_REWRITE mappings for "internal" channels. This includes the reprocessing channel which is difficult to test for any other way because it us designed to be invisible.

Support For the MT-PRIORITY SMTP Extension Defined in RFC 6710

Support for the MT-PRIORITY SMTP extension defined in RFC 6710 has been implemented. See the discussion about priority message handling in *Messaging Server System Administrator's Guide*.

New clonehosts Channel Option

A commonly requested capability is to clone all messages that meet some criteria and send them to an alternate destination. This can currently be accomplished in a variety of ways, including sieve capture, the `MESSAGE-SAVE-COPY` mapping, the `FORWARD` mapping, and various sorts of address rewriting tricks, and all of these mechanisms have different advantages and disadvantages.

In the specific case when the desire is to clone all messages sent to a particular channel to another channel while preserving the initial address that expanded to that channel, there is currently no way to get that specific effect. A capture action in a destination channel script can capture the message, but not the initial address. `MESSAGE-SAVE-COPY` has similar limitations and also requires playing queue management games.

The `clonehosts` channel option has been added to fill this gap. It accepts a single argument: A space-separated list of host names. Given a `clonehosts` setting of `host1 host2 host3` and a message sent to the addresses `initial1` and `initial2`, both of which expanded to one or more final recipient addresses destined to that channel, this setting will add the recipient addresses:

```
@host1:initial1
@host2:initial1
@host1:initial2
@host2:initial2
```

Use of Attributes Can Now Be Conditional on the Presence or Absence of Specific Object Classes

It is sometimes useful to be able to make use of certain attributes during address reversal and alias expansion conditional on the presence or absence of specific object classes on user entries. Among other things, this provides the means to support multiple schemata with conflicting attribute semantics simultaneously (assuming that the MTA implements the underlying semantics of the various attributes).

The MTA now supports this through the specification of object class qualifiers on attribute names specified in any of the `LDAP_*` MTA options controlling which attributes are consumed from user entries. For example, suppose you want to prevent the MTA from consuming the `mailDeliveryOption` attribute on selected entries, which will cause it to use the default, and you want to do this without actually removing the option from those entries. This can now be done by creating a new object class, for example `noDeliveryOption`, and specifying:

```
LDAP_DELIVERY_OPTION=mailDeliveryOption-noDeliveryOption
```

More generally, these options now accept up to four "positive" qualifiers (object classes which must be present before the option will be consumed) and up to four "negative" (object classes which must not be present before the option will be consumed).

These qualifications also apply to attributes consumed during address reversal. For example, suppose you wanted to use an object class to control whether or not the `cn` value on an entry is inserted as a personal name phrase. This could be done by defining a new object class, for example `cnAToPersonal`, and specifying:

```
LDAP_PERSONAL_NAME=cn+cnToPersonal
```

Note that this mechanism is not available for attributes in domain entries. Additionally this option does not work for the attributes that are used to actually look up user entries: `LDAP_PRIMARY_ADDRESS` (default `mail`), `LDAP_ALIAS_ADDRESSES` (default `mailAlternateAddress`), and `LDAP_EQUIVALENCE_ADDRESSES` (default `mailEquivalentAddress`).

New $]]]LDAP-URL[LDAP Query URL Mapping Template Substitution

LDAP query URL substitutions are normally of the form `$]LDAP-URL[]]LDAP-URL[` query form requires that a single result be produced. Multiple results cause the mapping entry to fail.

A `$]]]LDAP-URL[` form has now been added. This will cause all results returned by a successful query to be copied into the mapping result, separated by CRLF sequences.

New Flag Inputs to the AUTH_REWRITE Mapping

The following additional flags are now set as inputs to the AUTH_REWRITE mapping:

$A - Set if SASL has been used
$T - Set if TLS has been used
$P - Set if POP-before-SMTP was used
$E - ESMTP (EHLO as opposed to HELO) was used
$L - LMTP was used

Connection Retry After STARTTLS Negotiation Failure

If `maytlsclient` is in effect, the SMTP server offers STARTLS, and the negotiation fails, the SMTP client will now close the connection and try again on a new connection without attempting to negotiate TLS. In the past the SMTP client would abort the delivery attempt.

New receivedstate Channel Option

RFC 6729 specifies a new state clause for Received: header fields, which indicates the type of processing a message is about to undergo. The primary use for these clauses is to indicate when an operation is being undertaken that could cause a delivery delay, for example if the message has been quarantined.

A `receivedstate` destination channel option has been added to control the generation of state clauses. A single argument is required which specifies the state name to insert into the Received: fields of messages enqueued to the corresponding channel. The default, when `receivedstate` is not specified, is not to insert any state clause.

New implicitsaslexternal and explicitsaslexternal Channel Options

The `implicitsaslexternal` option on the current source channel causes the SMTP/SUBMIT server to perform an implicit AUTH EXTERNAL SASL operation when a MAIL FROM command is received provided the following conditions have been met:

- `mustsaslserver` is in effect and no authentication operation has been performed.
- An SSL/TLS layer has been successfully negotiated.
- The client provided a valid certificate as part of the SSL/TLS exchange.

  The `explicitsaslexternal` option, the default, disables this behavior.

The i;ascii-integer, i;ascii-casemap-collapse, and i;octet-collapse Comparators Now Supported by the MTA's Sieve Implementation

The `i;ascii-integer` comparator can be used to test signed or unsigned integer values.

The `i;ascii-casemap-collapse` comparator and `i;octet-collapse` comparator operate in generally the same fashion as the original comparators without the `-collapse` suffix, except that all folding white space characters (space, tab, carriage return, line feed) are removed from both the target and pattern strings prior to comparison.

New passsyntaxerrors and fixsyntaxerrors Source Channel Options

The `passsyntaxerrors` and `fixsyntaxerrors` source channel options have been introduced. `passsyntaxerrors` disables syntax error fix ups on address and message ID fields processed by the MTA. `fixsyntaxerrors` enables syntax error fix ups, and is the default.

Support Added for the foreveryport Sieve Extension

Support has been added for the `foreverypart` Sieve extension defined in RFC 5703. In addition to the

break control command, the implementation also supports the nonstandard continue control command:

```
continue [:name <string>]
```

Continue has the expected semantics: Control is passed to the bottom of the `foreverypart` loop.

Additional Settings for the LOG_USERNAME MTA Option

Bit 2 (value 4) of the `LOG_USERNAME` MTA option causes the primary mail address associated with the authenticated username to be logged. Setting bit 3 (value 8) causes the same address to be included in `LOG_ACTION` mapping probes. This address appears immediately after the username in both the log and in the mapping probe. If bit 4 (value 16) is set additional information associated with an authentication operation will be included in `U` records. If bit 8 (value 32) is set the same information appears in the `LOG_ACTION` mapping probe immediately after the authenticated username. At present this information is limited to the name of the authentication mechanism.

Support Added for the extracttext Sieve Extension

Support had been added for the `extracttext` Sieve extension defined in RFC 5703. Since Sieve is done as an overlay on top of the underlying language interpreter, the use of `extracttext` outside of a `foreverypart` is not detected as an error at compile time. Additionally, `extracttext` is only supported on leaf parts. It cannot be used on multipart and message/rfc822 parts.

Support Added to Sieve and Recipe Language for User-Defined Routines

Support for user-defined routines has been added to both Sieve and the Recipe language. Subroutine definitions have the general form:

```
sub routine-name {routine-body}
```

or if parameters are needed:

```
sub routine-name(parameter1, parameter2, ...) {routine-body}
```

The `return` control command can be used to return a specified result from the routine.

Parameters are passed by value and evaluation of parameters is lazy. If a parameter is never referenced it will never be evaluated. Note that evaluation of parameters in a particular order can be forced very easily:

```
sub f(p1,p2,p3) { p1; p2; p3; ... }
```

Local variables can be declared in a routine by specifying `my` control command immediately preceding the first use of the variable.

`autoincrement`, `autodecrement`, and the various augmented assignment operators are all allowed on parameters and local variables. So is the exchange operator `:=:`. However, exchange cannot be used with a global variable on the right hand side and a local variable or parameter on the left hand side.

For example, a factorial function can be defined as follows:

```
sub f(n) {if n <= 1 {return 1;} else {return f(n-1)*n;}}
```

Recursion is limited to 20 levels. A routine can only call itself recursively since there is currently no forward declaration mechanism.

An example of the use of `my` would be:

```
sub fib(n) {my s = [1, 1];
my a = 1;
my b = 1;
loop {exitif --n < 2; my c = a + b; s .= c; a = b; b = c;}
return s;}
```

Use of user-defined routines in Sieves is restricted to system-level Sieves.

Support Added for the Postfix's XCLIENT SMTP Extension

The Postfix's XCLIENT SMTP extension is now supported. The PostFix documentation for the extension can be found here:

http://www.postfix.org/XCLIENT_README.html

Use of XCLIENT is controlled by three source channel options: `noxclient`, `xclient`, and `xclientsasl`. `noxclient` is the default, and means that XCLIENT is not advertised in the response to EHLO and the XCLIENT command itself is disabled. If `xclient` is set the XCLIENT command is enabled and the NAME, ADDR, PORT, PROTO, and HELO attributes may be used. `xclientsasl` enables the LOGIN attribute in addition to all the others. It should be noted that LOGIN specifies an external identity that must then be bound to the session identity through the use of SASL EXTERNAL.

The primary visible effect of XCLIENT is on the contents of the Received: field the MTA adds. For example, if this XCLIENT command was executed:

```
xclient name=foo.example.com addr=1.2.3.4 helo=bar.example.com port=12345
```

It would result in a header of the general form:

```
Received: from bar.example.com (foo.example.com [1.2.3.4])
by example.domain.com (Oracle Communications Messaging Server 7u6-0.01
64bit (built Aug 18 2012)) with imapsubmit
id <01OJ9P51WPFC007KNZ@server.example.com> for user@example.com;
Mon, 20 Aug 2012 08:17:31 -0700 (PDT)
```

However, the ADDR and PORT attributes also change the contents of the transportinfo that appears in various mappings. Given the preceeding XCLIENT command, the transportinfo part of the mapping probes would change to something like:

```
TCP|<destaddr>|25|1.2.3.4|12345
```

BINARYMIME SMTP Extension Supported for Message Submission

The BINARYMIME SMTP extension defined in RFC 3030 is now supported for message submission. Binary messages submitted using this extension are immediately converted to regular 8bit MIME so the format of messages in the MTA queues is not affected, nor is the format of messages that are passed through the spam filter interface.

The `binaryserver` source channel option enables this extension. The `nobinaryserver` source channel option disables it, and is the default. The `binaryserver` source channel option enables the BDAT command for BODY=BINARYMIME messages even if `chunkingserver` is not in effect.

Security Enhancements to the BURL_ACCESS Mapping Table

$T in a BURL_ACCESS mapping makes use of TLS mandatory for the IMAP connection. $X disables use of TLS. $B in a BURL_ACCESS mapping disables certificate chain of trust validation for IMAPS: URLs and IMAP STARTTLS operations.

Enhanced Functionality for the $}...{ Mapping Metacharacter

The `$}...{` mapping metacharacter sequence has been enhanced to support user lookups in addition to domain attributes. The format is essentially the same as a domain attribute lookup `$}user,attribute{`.

The distinction lies in the attribute name; HULA attribute names always begin and end with a sharp sign #.

At present three user attributes are defined:

#canonical_user# - Canonical form of user identifier
#canonical_domain# - Canonical domain the user is associated with
#user_dn# - DN of the user's entry in LDAP

Enhancements to the MESSAGE-SAVE-COPY Mapping

The `MESSAGE-SAVE-COPY` mapping now provides a means of copying message files instead of or in addition to renaming them. If $G is specified, a file name is read from the mapping result and the current message file is copied there. If both $G and $Y are specified, the file is both copied and renamed. In this case the mapping result must be of the form:

copy-file-name|rename-file-name

If $Q is specified in addition to $Y an attempt will be made to tell the job controller to process the message file in its new location. $Q is intended to be used when a message file is moved from one queue to another.

$S can now be used in a `MESSAGE-SAVE-COPY` mapping to say that the message file has been renamed or otherwise processed by the mapping template and the file should simply be closed, not deleted or otherwise modified. $S is only effective if $Y is not specified.

New LOG_TIMES MTA Option

A LOG_TIMES MTA option has been added to the MTA. The default value is 0. If bit 0 (value 1) is set the time at which deferred delivery will take place is written to the transaction log immediately following tracking information. If bit 1 (value 2) is set the time at which deferred delivery will take place is included in the LOG_FILTER mapping probe immediately following tracking information.

Field-Specific Address Reversal and Personal Name Rewriting

Facilities have been added to perform field-specific address reversal and personal name rewriting.

Bit 12 (value 4096) of the USE_REVERSE_DATABASE MTA option will, if set, include the name of the header field the address being processed came from in the mapping probe, immediately after source channel, destination channel, and conversion flag information. A trailing colon is always included in the field name. A blank name appears when envelope addresses are being processed.

Bit 1 (value 2) of the USE_PERSONAL_NAMES MTA option causes the same header field name information described above to be included in the PERSONAL_NAMES mapping immediately following source channel, destination channel, and conversion tag information.

If bit 9 (value 512) of the INCLUDE_CONVERSIONTAG MTA option is set conversion tag information will be included in PERSONAL_NAMES mapping probes immediately following source and destination channel information..

If bit 2 (value 4) of USE_PERSONAL_NAMES MTA option is set any personal name generated during address reversal will not be used by default. Instead the revised name will be added to the PERSONAL_NAMES mapping probe immediately following the original personal name associated with the address. If the mapping wishes to use the revised name all it needs to do is return that name and set $Y.

New INCLUDE_SPARES2 MTA OPTION ADDED. INCLUDE_SPARES Renamed to INCLUDE_SPARES1

The INCLUDE_SPARES MTA option has been renamed to INCLUDE_SPARES1. An INCLUDE_SPARES2 MTA option has been added. It is a bit-encoded field similar to INCLUDE_SPARE1. The following table shows the bits, values, and their meaning.

Bit	Value	Meaning
0	1	Include LDAP_SPARE_1 attribute in FORWARD
1	2	Include LDAP_SPARE_2 attribute in FORWARD
2	4	Include LDAP_SPARE_3 attribute in FORWARD
3	8	Include LDAP_SPARE_4 attribute in FORWARD
4	16	Include LDAP_SPARE_5 attribute in FORWARD
5	32	Include LDAP_SPARE_6 attribute in FORWARD

The INCLUDE_SPARES option name has been retained as an alias for INCLUDE_SPARES1.

Bits 7 and 8 Added to USE_FORWARD_DATABASE MTA Option

Bit 7 (value 128) of USE_FORWARD_DATABASE will, if set, include the initial address presented for alias processing in the FORWARD mapping probe. Bit 8 (value 256) will, if set, include the current intermediate address in the FORWARD mapping probe. These addresses appear immediately before the final recipient address.

New headercut Destination Channel Option

A new destination channel option{{headercut}} has been added. If specified, headercut will cut the current message header down to no more than the specified number of bytes using a heurestic algorithm that removes or truncates header fields based on their relative importance. headercut requires a single

non-negative integer argument. A value of 0, the default, disables header cutting.

nonotify Sieve Action Added

A `nonotify` Sieve action has been added. This is similar to `novacation` except it suppresses all uses of the `notify` and `enotify` actions instead of `vacation`. Like `novaction` is can only be used in system-level Sieves.

In previous versions, `novacation` only took effect in Sieves evaluated after the one where `novacation` was invoked, and it affects all subsequent use of `vacation`.

In practice, `novaction` is used in the system Sieve to suppress user Sieve use of `vacation` and the system Sieve is evaluated before any user Sieves, so this had no significant effect. However, since such considerations may not apply for `nonotify`, both `nonotify` and `novacation` now only affect sieves attached to the same recipient address and evaluated later. This provides more consistent behavior.

Change to $N in the IP_ACCESS Mapping

A `$N` in the `IP_ACCESS` can now specify an argument which will be included in the diagnostic message logged in the result `R` log entry.

Correction to IP_ACCESS Mapping

The `IP_ACCESS` mapping has been corrected to actually include the channel the message is being dequeued from. Previously the mapping incorrectly included the name of the channel that enqueued the message.

Bit 1 (value 2) of the `USE_IP_ACCESS` MTA option, if set, will cause the the name of the channel that enqueued the message to be included in the probe.

$L in a URL Substitution Now Accepts an Integer Argument

Given $*n*`L`, *n* is a bit-encoded value. If bit 0 (value 1) is set the local part is dequoted prior to insertion. If bit 1 (value 2) is set a _, ~, or ` prefix, if present, will be removed. The default if n is not specified is 0.

Change to the Default Value of the Hold Delivery Option

Messages queued to the hold channel as a result of a `hold` `mailUserStatus` currently lose any attached subaddress. In order to prevent this, the default value of the hold delivery option has been changed from:

```
/hold=@hold-daemon:$A
```

to:

```
/hold=@hold-daemon:$1L+$2S@$D
```

Note that this setting takes advantage of the new change to $L in a URL substitution whereby it now accepts an integer argument. If subaddress preservation is needed in an earlier release an acceptable substitute is:

```
/hold=@hold-daemon:$U+$2S@$D
```

This will work correctly unless the address begins with a _, ~, or ` character.

Change to Logging of SPF HELO/EHLO Check Failures

SPF HELO/EHLO check failures now produce J records.

memcache Now Supported as a Place to Store Vacation Timeout Information

This is accomplished by specifying a memcache: URL in the VACATION_TEMPLATE MTA option. A typical setting would be:

```
memcache:///$U@$2I
```

in which case the memcache server is specified by the MEMCACHE_HOST and MEMCACHE_PORT MTA options. These options can be overridden by specifying the host and port in the URL as follows:

```
memcache://host:port/$U
```

If the port is not specified, it defaults to 11211, the usual port for memcache servers.

The content of the URL after the host and port specifies the key used to store information in memcache. The $U substitution provides the necessary information for the key, but a prefix or suffix can also be included if there is a need to distinguish the keys from other information stored in the memcache instance.

Connections to memcache are cached and reused. One final MTA option, MEMCACHE_EXPIRE, controls how old a cached connection can be and still be reused. It defaults to 5 minutes.

New $?whatever? URL Substituion

In URL substitution, a construct of the form:

$?*whatever*?

applies the store's hashdir algorithm to the string *whatever* to produce a directory path.

New SIEVE_RECEIVED MTA Option

A SIEVE_RECEIVED MTA option has been added to control whether or not a synthetic Received: field is added to the main message header prior to Sieve evaluation. A value of 1 (the default) enables the addition. 0 disables it.

Additional Settings for the USE_REVERSE_DATABASE MTA Option

Bits 13 (value 8192) and 14 (value 16384) in the USE_REVERSE_DATABASE MTA option have been defined so if they are set, they disable source block and recipient limit setting and capture actions based

on the envelope from (MAIL FROM) address (bit 13) and authenticated sender address (bit 14).

Timers Added to the MTA

Timers have been added to measure the total time the MTA spends waiting for various external components to return a response. All of the timers are controlled by the ENABLE_DELAY_TIMERS MTA option. Setting this option to 1 enables these timers. 0 disables them and is the default.

Timings are done on a per-message basis. The following timers are available:

1 - Time spent waiting for spam filters 1-8. (S1 - S8)
2 - Time spent waiting for mapping routine callouts. (MC)
3 - Time spent in the following specific mappings waiting for routine callouts:

PORT_ACCESS (PA)
FROM_ACCESS (FA)
ORIG_SEND_ACCESS (OSA)
SEND_ACCESS (SA)
ORIG_MAIL_ACCESS (OMA)
MAIL_ACCESS (MA)
AUTH_REWRITE (AW)
REVERSE (RV)
All Sieve mappings (S)

4 - Rewrite rule callouts (RR)
5 - Time spent creating an SMTP transaction for the MTA to process (STT)
6 - Time spent writing the queue file(s). (QW)

Note that timer 5 necessarily includes MTA processing time.

When logging this information it is formatted as follows (the field names are specified in the preceding list of timers):

```
S1,S2,S3,S4,S5,S6,S7,S8:MC,PA,FA,OSA,SA,OMA,MA,AW,RV,S:RR:STT,QW
```

Each value appears as an integer time in centiseconds followed by a semicolon and an integer use count:

```
T;U
```

Zero timer value/use count pairs are omitted entirely. Use counts of 1 are also omitted. In the specific case of spam filter, an outright filter failure is indicated by the presence of an F suffix followed by an integer code corresponding to the phase where the filter failed. Possible code values are:

Type of Filter Failure	Integer Code
LOG_SPAMFILTER_SYSTEM_FAILURE	1
LOG_SPAMFILTER_MESSAGE_FAILURE	2
LOG_SPAMFILTER_CONNECT_FAILURE	3
LOG_SPAMFILTER_MAILFROM_FAILURE	4
LOG_SPAMFILTER_AUTHADDR_FAILURE	5
LOG_SPAMFILTER_RCPTTO_FAILURE	6
LOG_SPAMFILTER_ENDRCPTTO_FAILURE	7
LOG_SPAMFILTER_HEADER_FAILURE	8
LOG_SPAMFILTER_ENDHEADER_FAILURE	9
LOG_SPAMFILTER_BODY_FAILURE	10
LOG_SPAMFILTER_ENDBODY_FAILURE	11
LOG_SPAMFILTER_SUBMIT_FAILURE	12

Finally, 0 elements of the comma-separated sublists may be truncated from the right.

Parsers should be aware that additional colon-delimited elements may be added to the list as a while, comma-separated elements to sublists, or even semicolon-separated elements to individual timer values.

The LOG_CALLOUT_DELAYS MTA option controls the logging of the above timer information. This option defaults to 0. Bit 0 (value 1), if set, causes the callout timing information to be logged immediately after delivery flags. The XML attribute name in XML format logs is cd. Bit 1 (value 2), if set, causes callout logging information to be included in the LOG_ACTION mapping probe, again immediately after delivery flags. In both cases the information is formatted as described above. Timings only appear in E (enqueue) log records.

Here are some actual logs entries illustrating what this looks like:

```
cd="1410,,,15:::123,25"
cd="1243,,,21:::127,33"
cd="172:1855;5,,107,248,400,500,600::4836,14"
cd="444,,,3F2:::63,43"
```

Support Added for the Sieve Duplicate Extension

Support for the Sieve duplicate extension specified in the Internet-Draft draft-ietf-appsawg-sieve-duplicate-00 has been implemented. There are two new MTA options associated with the implementation: MAX_DUPLICATES and DUPLICATE_TRACKING_URL. The former is an non negative interface that specifies how many duplicate tests may be performed in a single script. The default is 2.

DUPLICATE_TRACKING_URL specifies where duplicate tracking information should be stored. At present the value must be a memcache: URL of the form:

```
memcache://host:port/key-prefix
```

If the host isn't specified it defaults to the value of the MEMCACHE_HOST MTA option. It is an error for MEMCACHE_HOST not to be set in this case.

If the port isn't specified it defaults to the value of the MEMCACHE_PORT MTA option; if that option in turn isn't specified the default is 11211, the usual port for memcache servers.

Key-prefix, if specified, is prepended to the keys the duplicate extension sends to the memcache server.

Note that duplicate tests are performed during Sieve evaluation but no memcache updates are performed. It is only after the message has been successfully processed that updates are done.

Also note that duplicate information is implicitly qualified by the owner of the sieve. In the case of system-level sieves this will be the applicable postmaster address, so system-level sieves operate in shared namespace(s). Note that the :handle argument can be used to force system-level sieves to operate in their own namespace.

Support for Accessing Sleepycat Databases has been Removed from the MTA.

Note that the various ancillary utilities, in particular imsimta dumpdb, have not been removed so customers may continue to access any data they may have stored in existing MTA databases.

Additionally, facilities have been provided to use the memcache protocol as an alternative for direct use of Sleepycat. Note that memcachedb provides memcache protocol access to Sleepycat; it could be used to continue storing MTA information in Sleepycat, except with the advantage that multiple systems could share the same database.

The following MTA options control the use of the memcache protocol with various MTA databases:

GENERAL_DATABASE_URL General database
REVERSE_DATABASE_URL Reverse database
FORWARD_DATABASE_URL Forward database
DOMAIN_DATABASE_URL Domain database
ALIAS_DATABASE_URL Alias database
SSR_DATABASE_URL Server side sieve rules database

Each of these options can be used to specify a memcache URL of the form:

```
memcache://host:port/key-prefix
```

If the host isn't specified as part of the URL it defaults to the value of the MEMCACHE_HOST MTA option. It is an error for MEMCACHE_HOST not to be set in this case.

If the port isn't specified it defaults to the value of the MEMCACHE_PORT MTA option; if that option in turn isn't specified the default is 11211, the usual port for memcache servers.

Key-prefix, if specified, is prepended to the keys the duplicate extension sends to the memcache server.

The imsimta crdb utility has been extended to support loading data via the memcache protocol. This option is activated simply by specifying a memcache: URL instead of a destination file. A -timeout qualifier can be used to specify the timeout value to attach to the entries that are created.

The imsimta test -db utility can be used to test this new functionality in various ways. For example, assuming the GENERAL_DATABASE_URL MTA option is set to an appropriate memcache: URL, the following commands will test the ability to add, retrieve, and delete database entries.

```
% imsimta test -db -database=general
1000 entries processed, 1000 failures
% imsimta test -db -database=general -add
1000 entries processed, 0 failures
% imsimta test -db -database=general
1000 entries processed, 0 failures
% imsimta test -db -database=general -delete
1000 entries processed, 0 failures
% imsimta test -db -database=general
1000 entries processed, 1000 failures
```

This test uses an ascending sequence of entry values. Adding `-random=`*key* will use random hash values instead. `-repetitions` can be used to specify the number of test entries; the default is 1000.

New ACCESS_AUTH MTA Option

A new `ACCESS_AUTH` MTA option has been added. This is a bit-encoded field with two bits currently defined. If bit 0 (value 1) is set the value of the `SMTP AUTH` parameter is included in the `FROM_ACCESS` mapping probe immediately following the authenticated sender address. If bit 1 (value 2) is set the canonical username result produced by authentication is included in the `FROM_ACCESS` mapping probe immediately following the `SMTP AUTH` parameter. The default value for this option is 0.

$+C Flag in the FROM_ACCESS Mapping

The `$+C` flag, if set by `FROM_ACCESS` mapping, clears the "SASL used" flag for the remainder of the transaction.

New Logging for setconversiontag and addconversiontag Sieve Actions

The `setconversiontag` and `addconversiontag` Sieve Actions now log what conversion tag was set if the `LOG_FILTER` MTA option is set.

To test this:

1. Create a system Sieve of the form:

```
setconversiontag "whatever";
```

2. Set the LOG_FILTER MTA option to 1.
3. Send a message and the resulting E record in the log file should have a filter field containing the string:

```
setconversiontag "whatever"
```

In XML format logs, the quotes will appear as """.) And since there is a separate code path when messages go to the bitbucket, a message should be sent there. It should produce the same logging.

Maximum MTA Symbol Table Increased

The maximum MTA symbol table size, which applies to many things including general, forward, and

reverse text databases, has been raised from 2,000,000 to 5,000,000.

Nonstandard translactionlog Sieve Action Implemented

A nonstandard `transactionlog` Sieve action has been implemented for use in system-level sieves. The action takes a single string argument. All of the `transactionlog` actions in all of the Sieves applicable to a given user are concatenated into a single string which is intended to be stored in the transaction log.

The `LOG_TRANSACTIONLOG` MTA option controls the logging of the resulting string. This option defaults to 0. Bit 0 (value 1), if set, causes the string to be logged at the very end of enqueue (E) records. The XML attribute name in XML format logs is `tl`. Bit 1 (value 2), if set, causes the resulting string to be included in the `LOG_ACTION` mapping probe, again at the very end.

Note that a simple (and somewhat useful) test is to set `LOG_TRANSACTIONLOG` to 1 and set up the following system sieve:

```
if header :matches "subject" "*" {transactionlog "${0}";}
```

Transaction log enqueue entries will now contain the contents of the Subject: header.

Some TCP Channel Options Now Default to 1 Instead of 0

The following TCP channel options now default to 1 instead of 0:

`DISABLE_ADDRESS` (`XADR` command)
`DISABLE_STATUS` (`XSTA` command)
`DISABLE_GENERAL` (`XGEN` command)
`DISABLE_CIRCUIT` (`XCIR` command)

Setting these options explicitly to 0 in the channel options file will reenable the associated command.

New 64-bit Counters Added

A set of 8 signed 64 bit counters have been added whose values can be adjusted from Sieve scripts. These counters have no predefined meanings. They can be used for any purpose.

A nonstandard Sieve action `adjustcounter` has been added to manipulate these counters:

```
adjustcounter [:duplicate] [] counter [value]
```

`counter` specifies the counter to operate on. This must be an Integer in the range 1-8. `value` is the amount to adjust the counter by. If omitted it defaults to 1.

The counters associated with the current source channel are affected by default. The `:channel` nonpositional parameter can be used to switch to some other channel. Note that variable substitution can be used on the `<channel-string>` argument to select a channel computed by the script. Also note that the channel must be defined in the configuration. Arbitrary channel names are not allowed.

`adjustcounter` can only be used in system-level Sieves. An error will occur if an attempt is made to use it from user-level Sieves.

Sieve scripts may be reevaluated multiple times, for example when a message is sent to multiple recipients and the script employs an envelope "to" test. When this happens, it is normally not desirable for the counter operation to be repeated, so counter adjustments are suppressed by default when scripts are reevaluated. This default can be overridden by specifying the `:duplicate` nonpositional parameter.

The counters show up in `imsimta counter -show` output as follows:

```
Sieve counter [1] 1
Sieve counter [2] 10
Sieve counter [3] 10
Sieve counter [8] -15
```

Note that counters can have negative values. Also note that counters with a value of 0 are suppressed from the display.

These counters can also be retrieved through the `PMDFgetChannelCounters64` in the PMDF API.

Nonstandard `setenvelopefrom` Sieve Action Implemented

A nonstandard `setenvelopefrom` Sieve action has been implemented:

```
setenvelopefrom <new-envelope-from-string>
```

As the name implies, `setenvelopefrom` overrides the message's envelope from field with the specified value.

`setenvelopefrom` can only be used in system-level Sieves. An error will occur if an attempt is made to use it from user-level Sieves.

New LDAP_GROUP_LAST_ACCESS_TIME MTA Option

Support has been added to keep track of the last access time for email groups defined in LDAP. This is accomplished by setting the new MTA option `LDAP_GROUP_LAST_ACCESS_TIME` to the name of an LDAP attribute. If this attribute is present in a group's LDAP entry, the MTA will update the attribute each time the group is successfully accessed for purposes of sending mail or expanding a mailing list. RFC 3339 format is used, e.g., "2013-09-29T17:38:52Z".

In order to prevent excessive LDAP writes, the attribute is read prior to writing and a write is only done if the current time exceeds the stored time by at least 30 minutes. A write is also done if the attribute does not contain a valid RFC 3339 time, making it possible to set the initial value to something like "<never accessed>".

Acceptance of Illegal Date: Header Fields

Date: header fields with values of the general form:

```
Thu, 10 Oct 2013 16:37:39 \-04:00
```

are showing up. This format is technically illegal - colons are not allowed in the time zone value - but the meaning of such values is obvious. Accordingly, date-time parsing heuristics have been adjusted to

support such values. Additionally, when a syntactically illegal date-time value is found that cannot be parsed, in addition to treating it as GMT (+0000), the invalid string will be inserted as a comment, for example something like:

```
Thu, 10 Oct 2013 16:37:39 \-9999
```

will become:

```
Thu, 10 Oct 2013 16:37:39 \+0000 (-9999)
```

Nonstandard XUNAUTHENTICATE Extension and XUNAUTHENTICATE Command Added

A nonstandard `XUNAUTHENTICATE` extension and an associated `XUNAUTHENTICATE` command have been added to the SMTP server. This command is only valid after successful authentication has been performed, and the capability only shows up in the EHLO response at this point as well. Successful execution of this command will return the session to an unauthenticated state. Any channel switching done by `saslswitchchannel` will also be undone. There are no options to enable or disable this command.

draft-ietf-appsawg-rrvs-header-field-01 Implemented

draft-ietf-appsawg-rrvs-header-field-01 has been implemented. The following table shows the new options and parameters that have been added and their descriptions:

Option or Parameter	Description
`ignorerrvs` source channel option	Ignore Require-Recipient-Valid-Since: header fields. Do not offer or accept the RRVS SMTP extension. This is the default.
`checkrrvs` source channel option	Check Require-Recipient-Valid-Since: header fields enable support for the RRVS SMTP extension.
`LDAP_DOMAIN_ATTR_CREATION_DATE` MTA option	Specifies the name of an LDAP attrible for domain entries where domain creation date information is stored. The information must be in RFC 3339 format (a superset of the format used for vacation start and end times).
`LDAP_CREATION_DATE` MTA option	Specifies the name of an LDAP attribute for users and groups where account creation date information is stored. The information must be in RFC 3339 format (a superset of the format used for vacation start and end times).
{{[creation_date	creation_date]}} nonpositional alias parameter Used to specify the creation date for the alias, again in RFC 3339 format.

New DEFER_HEADER_ADDITION MTA Option

A `DEFER_HEADER_ADDITION` MTA option has been added. This option defaults to 0 so the default behavior will be for Sieves to see added headers on redirected messages. Setting the option to 1 restores the old behavior.

Change to the mgrpAddHeader Attribute

The `mgrpAddHeader` attribute (or whatever attribute is specified by the `LDAP_ADD_HEADER` MTA

option) is now implemented on top of the header addition rather than the header trimming facility. This means that added headers will be seen by list recipient Sieves even if they are executed as part of the same enqueue operation.

New CONNECT_TIMEOUT spamfilter Plugin Option

A `CONNECT_TIMEOUT` option has been added to the milter, SpamAssassin, ClamAV, and ICAP spamfilter plugins. This option is used to specify the time, in seconds, to wait before giving up on a connection to the filtering server.

If `CONNECT_TIMEOUT` is not specified, it defaults to the plugin's `TIMEOUT` setting if it has one. If neither option is set, `CONNECT_TIMEOUT` defaults to 60 seconds.

The default for the `TIMEOUT` setting has also been changed to 120 seconds for all of these plugins.

New LOG_FUTURERELEASE MTA Option

See the discussion about Message Tracking and Recall in *Messaging Server System Administrator's Guide*.

New Qualifiers for imsimta test -expression and imsimta calc

`imsimta test -expression` and `imsimta calc` now accept two new qualifiers:

`-sender=`*authenticated-sender* Set the authenticated sender address.
`-username=`*username* Set the authentication username.

Note that these qualifiers simply simulate the setting of these fields. They do not actually perform any sort of authentication operation.

Sieve Variable and Encoded Character Substitutions Can Now be Performed on Nonstandard Functions

If enabled, Sieve variable and encoded character substitutions are performed on the arguments to all Sieve tests and actions. However, such substitutions are not performed on nonstandard built-in functions like `left$`, `right$`, `encode`, `decode`, and so on.

It is sometimes convenient to perform such substitions on function arguments, in particular to get at the match result variables `${0}`, `${1}`, an so on. A new function `$( )` has been added to do this. The following script fragments are now equivalent:

```
require ["variables"];
...
if header :matches "foo" "*" {
set "a" "${0}";
...

require ["variables"];
...
if header :matches "foo" "*" {
a = $("${0}");
...
```

Support for draft-delany-nullmx-01.txt Implemented

Support for draft-delany-nullmx-01.txt has been implemented: A domain with a null MX record entry, for example:

```
nomail IN MX 0 .
```

will be detected when the MX is resolved and treated as an invalid destination host and bounce the message.

There is presently no option to disable this behavior in the SMTP client.

An additional bit has been defined in `returnenvelope` to enable a similar check for domains being tested as valid sources of mail. If bit 6 (value 64) of `returnenvelope` is set in addition to setting bit 3 (value 8), a null MX domain will be rejected as invalid in MAIL FROM addresses.

MeterMaid's Client check_metermaid.so Now Supports Multiple MeterMaid Servers and SSL for Communication

If you have multiple MeterMaid tables in your deployment, you may be able to improve overall performance by distributing them across multiple MeterMaid servers. The check_metermaid.so client supports associations between MeterMaid tables and the servers that are responsible for their respective tables. The client also supports per server options for concurrency and use of SSL. For additional information, see the discussions about configuring `check_metermaid.so` clients to access multiple MeterMaid servers and using and configuring MeterMaid for access control in *Messaging Server System Administrator's Guide*.

New affinitylist Channel Option

The `affinitylist` channel option has been added to the `mx` channel option group. When a channel is marked with the `affinitylist` option, MX lookups are disabled and the specified host is looked up using the affinity subsystem instead. This option is intended for use on LMTP channels delivering to a group of message store back ends with affinity support enabled.

A get_path(r) Function Added to Recipe Language

A `get_path`(r) function has been added to the recipe language which returns a file path to the current server instance. *r* should be one of:

"server" - return path to the server root
"data" - return path to the data root
"config" - return path to the configuration root

Note that having these functions means there's a simple command in `msconfig` that can be used to access these values:

```
msconfig> exec get_path "server"
> "/opt/sun/comms/messaging64"
msconfig> exec get_path "config"
> "/opt/sun/comms/messaging64/config"
msconfig> exec get_path "data"
> "/opt/sun/comms/messaging64/data"
```

Message Tracking and Recall

A general message tracking and recall facility has been implemented. See the discussion about message tracking and recall in *Messaging Server System Administrator's Guide*.

New Nonstandard Sieve Override

Multiple Sieves can be and often are evaluated for a given message recipient. When this happens some actions, such as `addheader`, `capture`, or `addconversiontag`, are accumulated across all applicable Sieves. But actions that determine the ultimate disposition of a message have to come from a single Sieve.

The general rule is that the most specific Sieve that sets a disposition wins. So if, for example, a `keep` done by a system Sieve would be overridden by a `discard` done by a user Sieve.

A couple of nonstandard Sieves have previously been added to override this determination order. In particular, the most general Sieve that performs a `jettison` or `refuse` action will determine the disposition of a message for a given recipient.

The use cases for `jettison` and `refuse` are obvious. If system policy has determined that a user cannot see a message, the user's own preferences need to be overridden. The `jettison` and `refuse` actions also have the advantage that they are incompatible with other disposition actions, making them easy to use.

In contrast, the other disposition actions (`keep`, `discard`, `fileinto`, `redirect`, etc.) are all compatible with each other, and in practice are used in a wide variety of combinations and permutations. So separate `override` variants of these actions don't make sense.

Recently, a need has surfaced for a system Sieve to perform source routing using `redirect` and possibly `fileinto`. For this to work correctly, such Sieves need to be able to override disposition actions specified by more specific Sieves. Accordingly, a nonstandard Sieve `override` action has been added. A Sieve that uses this action becomes the Sieve determining the disposition of the message. If multiple Sieves employ `override`, the most general one wins.

For example, suppose you have a system Sieve:

```
redirect "foo@bar";
```

And the recipient has a user sieve:

```
keep;
```

The `keep` action wins and the message is filed into the recipient's INBOX folder. Now suppose the system Sieve is changed:

```
require "override"; override; redirect "foo@bar":
```

Now the `redirect` action wins and the message is redirected.

A :header Argument Can Now be Used with capture :dsn and capture :journal

If specified, the resulting capture message only contains the header of the original message, not the body. (Note that when this is done with journal format the resulting message does not and cannot comply with Microsoft's Exchange journal format standard.)

The CAPTURE_FORMAT_DEFAULT MTA option also accepts two new values. The following table shows the acceptable values and of the CAPTURE_FORMAT_DEFAULT MTA option and their descriptions.

Value	Description
0	Full message capture, DSN format
1	Full message capture, raw message
2	Full message capture, Exchange journal format
4	Header capture, DSN format
6	Header capture, Exchange journal format

Finally, LDAP attribute tags can be used to select what type of capture is done. This set of tags is now expanded to:

```
;format-message (0)
;format-report (1)
;format-journal (2)
;format-report-header (4)
;format-journal-header (6)
```

Largest Token in Command Line Reader Increased

The size of the largest token in the command line reader used by several utilities, notably `msconfig`, has been increased from 200 to 1000 octets.

New ALLOW_SESSION_BLOCKS and ALLOW_TRANSACTION_BLOCKS SMTP Channel Options

Two new SMTP channel options have been added:

`ALLOW_SESSION_BLOCKS` - Imposes a block limit on SMTP server session as a whole. The session is disconnected if this limit is exceeded.

`ALLOW_TRANSACTION_BLOCKS` - Imposes a block limit on any transactions handled by the SMTP server. The session is disconnected if this limit is exceeded.

New destinationpassthrough Channel Option

The `passthrough` channel option, if specified on a source channel, disables header rewriting and address reversal as well as any charset conversions. Since this is a source channel option, it applies to all recipients of a given message.

A `destinationpassthrough` channel option has been implemented to allow more selective use of `passthrough` mode. Any message enqueue not to a destination channel marked with the

destinationpassthrough option will be done in passthrough mode as long as the source channel isn't itself marked with the submit channel option. If submit mode is engaged the, destinationpassthrough channel option is a no-op.

$O Flag in AUTH_ACCESS Mapping

Setting the $O flag in the AUTH_ACCESS mapping causes nonauthoritative "no such domain" (NXDOMAIN) errors from the DNS to be treated as temporary failures.

Important note: Modern DNS behavior is such that whether or not the authoritative bit is set in the result no longer means much. As such, setting this flag unnecessarily can lead to backlogs of messages to invalid domains in the queues. However, there may be some obscure corner cases where it's helpful, and even if it isn't, having the ability to do this lets sites try it and see for themselves whether it helps.

New Functionality for Sieve reject, ereject, and refuse Actions

The Sieve reject, ereject, and refuse actions will now parse any extended SMTP error code (e.g., "5.7.2") that appears at the beginning of the action's string argument in a system-level script. This capability is not available to user-level Sieves.

New setoperation Sieve Action

A new {{setoperation} system-level Sieve action has been added. This action can be used to override the type of enqueue operation being performed. Note that this can be done on a per-recipient basis.

setoperation accepts a single string argument specifying the operation type. The possible operation types are:

```
"relay"
"submit"
"passthrough"
"default"
```

Note that since setoperation is a Sieve action, any effects it has only affect MTA behavior after Sieve evaluation. In particular, initial header fixups done on receipt of the header occur before Sieve evaluation and are therefore unaffected by this action. Only the passthrough channel option affects initial header fixups.

Additional Bit Defined in the spamfilterX_received Set of Options

An additional bit in the spamfilterX_received set of options has been defined. Bit 2 (value 4) is used to specify that the header NOT be generated during reprocessing operations. (This is useful since the enqueue to the reprocess channel already added a Received: header field.)

New Sieve Warning Facility

Warnings that occur during sieve evaluation will now result in a warn clause in log_filter results. At present this includes memcache protocol issues and issues with the duplicate and vacation extensions.

Additionally, a warn action has been added to the language. It takes either a string or a list as an argument, which is then included in the warn clause in log_filter results.

Support for draft-ietf-appsawg-email-auth-codes-07 Implemented

This consists of changing the response code for a MAIL FROM address not in the DNS from 5.1.8 to 5.7.25.

Milter Extension to Allow Per-Recipient Modification Actions

A Milter extension to allow per-recipient modification actions has been implemented. See the discussion about using Milter in the spam and virus filtering section of *Messaging Server System Administrator's Guide*.

UNAUTHENTICATE Command Disabled by Default.

The UNAUTHENTICATE command is now disabled by default. It can be enabled by setting `imap.capability_x_unauthenticate` to 1 (or `service.imap.capability.x_unauthenticate` for legacy configuration).

Messaging Server Supports IMAP LIST Extension For Special-Use Mailboxes

Messaging Server now supports the IMAP LIST extension for special-use mailboxes as defined in RFC 6154. This enables compliant mail clients to identify (and label) the folder used for Trash, Drafts and other special uses regardless of the user's language or other name variations.

Conversion Channel Enhancements

Messaging Server by default does not create environment variables containing untrusted information that could trigger the "shellshock" bug in bash. Messaging Server also doesn't use system() and similar calls to execute programs. However, the conversion channel is highly flexible and could be configured by customers in a such a way that the resulting configuration would be vulnerable.

Accordingly, the following enhancements have been made:

- Environment variables created by the conversion channel are checked for the prefix "() {" that is the trigger bash looks for to parse function definitions out of environment variables. The environment variable is not created if this prefix is found.
- A new set of conversion options have been added:
  - `illegal-values` - Specifies a list of glob-style wildcard patterns. Each prospective environment variable containing message data is checked against this list and won't be created if a match is found. The first character of this option's value is taken as the separator for the pattern list. For example, in order to specify the patterns "{}*" and "*(*)*", you could write:

    ```
    illegal-values=,{}*,*(*)*
    ```

  - `illegal-chars` - Specifies a list of characters. Each prospective environment variable containing message data is checked against this list and won't be created if any character on the list is found.
  - `remove-chars` - A list of characters which, if found, will be removed from environment variable values.
  - `quoting-chars` - A character string used to quote any of the characters specified by the `quote-chars` option.
  - `quote-chars` - A list of characters which, if found, will be prefixes by the string specified by `quoting-chars`.

The following environment variables created by the conversion channel are affected by these options:

- `INPUT_TYPE`, `INPUT_SUBTYPE`, `INPUT_DISPOSITION`, `INPUT_DESCRIPTION`,

INPUT_LANGUAGE, OUTPUT_TYPE, OUTPUT_SUBTYPE, OUTPUT_DISPOSITION, OUTPUT_DESCRIPTION, and OUTPUT_LANGUAGE.

- Any environment variable created by a PARAMETER-SYMBOL-n option.
- Any environment variable created by a DPARAMETER-SYMBOL-n option.

MTA's Internal Buffers Increased

The MTA's internal buffers for building and processing LDAP URLs related to alias expansion and address reversal have increased in size from 1024 to 4096 octets.

nonlocal mailDomainStatus Setting

Setting the mailDomainStatus attribute for a domain to nonlocal tells the MTA to treat the domain as non-local:

- $V options in rewrite rules will fail.
- Alias lookups will fail.
- Address reversal operations will be aborted after the domain lookup.

Note that unlike setting the mailDomainStatus to "unused", domain-level =attributes such as block limits and overriding postmaster addresses will be applied.

New Sieve Environment Item vnd.oracle.tracking-id

This item returns the tracking identifier for the current message. The query will fail if there's no tracking ID.

New Sieve Environment Item vnd.oracle.message-hash

This returns any message calculated by the generatemessagehash channel keyword. The query will fail if no message hash was generated.

New LDAP_DOMAIN_ATTR_SUBADDRESS MTA Option Added

This option specifies a domain attribute which is then used to indicate whether or not address lookups associated with this domain should accommodate subaddressing. Specifically, setting this attribute to No, 0, or false will limit lookups to the full address and not attempt to lookup, say, foo@example.com in addition to foo+bar@example.com.

New -source=source-channel and -rsecret=recall-secret Qualifiers

These qualifiers have been added to imsimta calc and imsimta test -expression to specify the source channel and recall secret.

New USE_TEMPORARY_ERROR MTA Option

A new MTA option USE_TEMPORARY_ERROR has been added that controls whether or not certain errors returned by the MTA are marked as permanent or temporary. Each bit in this option corresponds to a specific error condition and when set instructs the MTA to return a temporary error. The default value for this option is 0. The following table shows the defined bits, their values, and the errors.

Bit	Value	Error
0	1	user disabled; cannot receive new mail
1	2	alias/group disabled; cannot receive new mail
2	4	unknown or illegal alias
3	8	unknown host or domain
4	16	unknown or illegal user

Additionally, source channel options `usetemporaryerror` and `usepermanenterror` have been added that provide functionality equivalent to the `USE_TEMPORARY_ERROR` and `USE_PERMANENT_ERROR` MTA options on a per-channel basis. Both options require a single nonnegative integer argument, and if specified override the corresponding MTA option.

New Channel Counters for Authentication Successes and Failures

Channel counters for authentication successes and failures have been added. These appear in `imsimta counter -show` output as:

```
Successful authentications 7
Failed authentication attempts 0
```

Enhanced imsimta test -rewrite Output

`imsimta test -rewrite` output of rewriting errors has been enhanced in some cases to include an indication of whether the error is classified as temporary (4yz) or permanenent (5yz) by the MTA.

imquotacheck -p <partition> Option to Report Quota for the Specified Partition.

This option can either be used with or without `-a` option.

imcheck -B <bound> Option Added

`imcheck -B <bound>` option has been added to use the user specified delimiter while dumping `mboxlist` databases.

The option is used only when dumping a database like `imcheck -d db_name [-B <bound>|]`. The given bound can be any string.

mboxutil -B <bound> Option Added

`mboxutil -B <bound>` option has been added to use the user specified delimiter while listing the mailboxes.

The option is always used with `-l` option like mboxutil `-l [-B bound]`. The given bound can be any string.

imsconnutil -k Option Added

`imsconnutil -k` option provides feedback about whether the users given as input are killed.

If the UID is provided and the user doesn't exist, it provides feedback that the user is invalid. If the user is not connected, it says that the user is not connected. If a user cannot be killed, it says that the user cannot be killed.

imsbackup Reports the Amount of Data Backed Up Per Mailbox in Verbose Mode

When used in verbose mode, `imsbackup` reports the amount of data backed up per mailbox in bytes.

imsbackup Gives the Progress Information About Backup

During backup in partial mode (it has a start date), `imsbackup` reports the progress in terms of mailboxes backed up (i.e. progress in terms of percentage of mailboxes backed up to the total mailboxes needed to be backed up). During backup in full mode, `imsbackup` reports the progress in terms of links/messages backed up (i.e. progress in terms of percentage of links/messages backed up to the total links/messages needed to be backed up).

jmqnotify Expects Acknowledgement by Default for Persistent Messages

`jmqnotify` expects an acknowledgement for messages, produced by a plugin that has the Delivery Mode as persistent, when received by the JMQ Broker. When these persistent messages cannot be sent to the JMQ Broker, they are added to the dropped count which is periodically reported to the error log. The plugins which have Delivery Mode as non-persistent do not expect an acknowledgement. These non-persistent messages are not reported to the error log when they cannot be delivered to the JMQ Broker.

ensnotify Has diskquota and diskquota Parameters in Messages

`diskquota` and `diskquotaUsed` attributes are added for notification messages from append, expunge, overquota, underquota, delete operations. All instances of the parameter, `diskquotaUsed` is changed to `diskquotaused` in the messages from ENS.

Text Message Preview Cache

If the message body type is TEXT/PLAIN, we now save the first N bytes of the message body in the cache file for fast preview access. To do this, we added a new configuration option `store.cachepreviewlen` to configure the number of bytes to save in the cache file.

Message Store Transaction Logging

We have implemented message store transaction logging. For details, see the discussion on implementing and configuring message store transaction logging in the Messaging Server Unified Configuration Administration Guide.

Log Rollover Manager

Transction, IMAP and POP logs are managed by a separate rollover daemon that supports both time based rollover or size based rollover. The rollover manager runs as service which can be started and stopped with `start-msg` and `stop-msg`. The time and size limits can be set through the existing logfile configuration.

SSL Support Added to IMAP Search When Communicating with Index and Search Service

SSL support has been added to the IMAP server when it communicates with Indexing and Search

Service to send and receive search requests.

To enable this feature, a new option has been added:

`imap.indexer.sslusessl` (Unified Configuration)
`service.imap.indexer.sslusessl` (legacy configuration)

If set to 1, then the IMAP server uses SSL to authenticate to Indexing and Search Service, using the `imap.indexer.port} option (Unified Configuration) or {{service.imap.indexer.port` option (legacy configuration) introduced in Messaging Server 7.0.5.0.31.

The description of the `imap.indexer.port` option in Unified Configuration or the `service.imap.indexer.port` option in legacy configuration has been updated:

Description: The port Indexer option specifies the TCP port on which ISS listens for incoming TCP connections, i.e., the TCP port to which Messaging Server should connect to communicate with ISS. If the `service.imap.indexer.sslusessl` option is set, the IMAP server uses SSL to authenticate to ISS on this port.

Charset Name IMAP-MODIFIED-UTF-7 replaced by IMAP-mailbox-name

When IMAP mailbox names contain non-ASCII characters, they are encoded in a special format described in RFC 3501 section 5.1.3. See http://tools.ietf.org/html/rfc3501#section-5.1.3 for more information.

We had previously used the name `IMAP-MODIFIED-UTF-7` for this encoding, but have switched to the de-facto name for this encoding used by the ICU library: `IMAP-mailbox-name`.

Added checksum Field to the Mailbox Header

We have added a `checksum` field to the index header (`mboxidxhdr`) to validate the mailbox header. The checksum is a CRC32 of all the other fields in the header.

ENS Generates a New Notification on Folder Rename Event

Whenever a folder is renamed, a new ENS notification is generated. There is no configuration option to enable/disable the generation of events on rename operation. The parameters or arguments in the event generated are same as the JMQ notification generated on Folder Rename event.

Changes to IMAP APPEND

IMAP APPEND can now reject large messages. A new `maxmessagesize` Unified Configuration option has been added to reject large messages appended to the mailbox. It specifies the maximum message size that IMAP clients are allowed to save via the IMAP APPEND command. The default is 4294967295.

To set the option to 5000000, for example, run the following command:

```
msconfig set imap.maxmessagesize 5000000
```

Running the `msconfig show` command on the option will show the setting.

```
msconfig show imap.maxmessagesize
role.imap.maxmessagesize = 5000000
```

An attempt to send a message exceeding the specified size will result in the following error message.

```
A NO [TOOBIG] Message too large
```

Additional changes to IMAP APPEND will have the following effects:

- Customers will see fewer `mailbox locked` errors that cause delivery delays.
- APPEND will spool messages in transit to a new `append_temp` directory in each partition. If this transfer is interrupted, this will be cleaned up later by `imexpire`. On success it will be hard-linked into the user's mailbox.
- It will be possible to have multiple append commands in progress to the same mailbox. This was not previously possible.
- The I/O cost of doing an IMAP append will increase slightly due to the additional hard-link operation.

NSS version check

The `imsimta version` command now displays the version of NSS installed. An example of an NSS version is:

NSS Library Version: 3.17.4 Extended ECC

It can also display a warning like this example:

The NSS library version (3.13.1 Extended ECC) appears to be less than the minimum recommended NSS version (3.17.4). Please check the online documentation to see if an NSS upgrade is recommended.

And if the NSS version gets too outdated it can display a message similar to the following:

ld.so.1: nssversion: fatal: libnss3.so: version 'NSS_3.13' not found (required by file /opt/sun/comms/messaging64/lib/nssversion) ld.so.1: nssversion: fatal: libnss3.so: open failed: No such file or directory

Change to the Default SSL/TLS Cipher Suites

The following cipher suite is no longer enabled by default starting with this release of Messaging Server.

- `SSL_RSA_WITH_RC4_128_MD5`

The following cipher suites are enabled by default starting with this release of Messaging Server:

- `TLS_RSA_WITH_AES_256_CBC_SHA`
- `TLS_RSA_WITH_AES_128_CBC_SHA`

These default changes are the opposite of the defaults in previous releases of Messaging Server. If you are using a mixture of old and new servers, it is recommended you also enable these two cipher suites in Messaging Server 7 Update 5 and prior releases with the `ssladjustciphersuites` option for unified

configuration or the `local.ssladjustciphersuites` `configutil` parameter for legacy configuration. Otherwise a slower cipher suite, such as `SSL_RSA_WITH_3DES_EDE_CBC_SHA` may be used when SSL connections are made between versions.

This information is now included in the following logs:

- Protocol log at `info` log level
- Protocol transcript, if enabled
- `msgtrace` log
- POP mailbox status log

The POP log now includes the `authtype` and `auth` session ID.

STARTTLS Option for All LDAP Connections

When the `base.ldaprequiretls` option is set to 1, then connections to LDAP that are not otherwise over LDAPS (port 636) will use the LDAP StartTLS control to negotiate TLS protection. This option is only available in Unified Configuration mode.

SSLv3 Disabled By Default

The `sslv3enable` option now defaults to 0 instead of 1. This may cause interoperability problems with third party products that have TLS 1.0 disabled by default but have SSL 3.0 enabled. Such products may have security vulnerabilities and may need to be updated for security reasons.

immonitor-access Has SSL and SASL Support

The immonitor-access tool has SSL/SASL support. Users can add the `-X` switch to enable SASL or the `-T` switch to enable SSL.

Changes to restricted.cnf, the Pipe Channel, and Privileged Shared Libraries.

The following changes are related to Unix user identity that improve product security.

- `restricted.cnf` is now required by default.
- Pipe channel user switching is now disabled by default.
- Privileged shared libraries must be owned by `root` or `bin`.

Compatibility impact summary:

- Pipe and native channels are not authorized to run programs as other users by default. Set `allow_pipe_setuid` to 1 in `restricted.cnf` owned by root to restore the behavior of previous releases.
- Customer-provided shared library plugins that run with privilege, particularly those called from `PORT_ACCESS` by the dispatcher, must now be owned by root or bin to operate.
- `IMTA_USER` `imta_tailor`, `local.serveruid` `configutil` option and MMP `BeTheUser` options are now are ignored. Only the `restricted.cnf` `user` option determines the userid for server processes.
- The dispatcher `user` option is now ignored.
- The `IMTA_USER_USERNAME` `imta_tailor` option is now ignored, but will be copied to the `restricted.cnf` `noprivuser` option on upgrade if that file did not previously exist. This will default to the `nobody` Unix user ID unless `restricted.cnf` is owned by root and the `noprivuser` option is set to the desired Unix user ID.
- The pipe channel `user}` `channel keyword is deprecated and replaced by the` `{{pipeuser` option in `restricted.cnf`.
  The `configutil -o local.serveruid` command will always show the value of the user

option in `restricted.cnf`.

In Messaging Server 7.0.5, the `restricted.cnf` file had to be owned by "root". In Messaging Server 8.0, that is still recommended, but the `restricted.cnf` file can also be owned by the user identity listed in the file. For example, if `restricted.cnf` contains `user=mailsrv` then the `restricted.cnf` file can be owned by the mailsrv user and the product will operate. When upgrading a legacy configuration to Messaging Server 8.0, the product will automatically create a `restricted.cnf` file based on the `imta_tailor` file if one does not already exist. The new `restricted.cnf` file will inherit permissions from `imta_tailor`. If the configure utility has been run since using a version starting with Messaging Server 7.0.5, even in legacy configuration mode, then a `restricted.cnf` file owned by root will already exist. Note that all options in `restricted.cnf` other than the `user` option will be ignored unless `restricted.cnf` is owned by root.

The pipe channel (and the rarely used native channel) allow Messaging Server to perform legacy Unix-style delivery to files or other processes. These other processes may run with user identities other than the default mailsrv user. In previous releases, the ability to do this was enabled by default. Starting with Messaging Server 8.0, this capability is disabled by default to improve security. The pipe channel can be authorized to run software as a single user other than mailsrv by setting the `pipeuser` option in a `restricted.cnf` file owned by root. This is recommend if possible. The pipe and native channels can be authorized to run as any user identity by setting the `allow_pipe_setuid` option to 1 in a `restricted.cnf` file owned by root.

Certain server processes run with elevated privilege, including the pipe channel and dispatcher use of the `PORT_ACCESS` mapping. These processes have the ability to load shared libraries for purposes such as mapping table callouts. There is now a requirement that shared libraries run by privileged server processes must be owned by root or bin. Customers who have built their own shared library callouts may need to set the permissions. This is most likely to impact customers using the `PORT_ACCESS` mapping. As the SMTP server is not privileged, most mapping callouts are not impacted by this change.

The dispatcher "user" option has been ignored since around 200 ,so the reference documentation has been updated to reflect this change.

Bundled NSS Upgraded to NSS 3.17.4

This release of Messaging Server upgrades NSS to version 3.17.4. Previously we supported SSL 3.0 and TLS 1.0 only. This adds support for TLS 1.1 and TLS 1.2. There is a new option to enable TLS 1.2 (base.tlsv12enable). TLS 1.2 is off by default (as designed by NSS developers).

Message Store Manual Failover

We have implemented message store manual failover. For additional information, see the discussion about message store manual failure in *Messaging Server System Administrator's Guide*.

Chapter 10. Configuring and Administering Multiplexor Services

Configuring and Administering Multiplexor Services

This information describes the Messaging Multiplexor (MMP) for standard mail protocols (POP, IMAP, and SMTP). Previous releases also described the Messenger Express Multiplexor used for the Messenger Express web interface, but beginning with **Messaging Server 7**, this is no longer needed. See To Configure HTTP Services.

Topics:

Multiplexor Services

A multiplexor is necessary to achieve horizontal scalability (the ability to support more users by adding more machines), because it provides a single domain name that can be used to connect indirectly to multiple mail stores. A multiplexor can also provide security benefits.

While MMP is managed separately from Messaging Server, the Messenger Express multiplexing is built-in to the HTTP service (`mshttpd`) that is included with the Message Store and Message Access installation.

Multiplexor Benefits

Message stores on heavily used messaging servers can grow quite large. Spreading user mailboxes and user connections across multiple servers can therefore improve capacity and performance. In addition, it can be more cost-effective to use several small server machines than one large, high-capacity, multiprocessor machine.

If the size of your mail-server installation requires the use of multiple message stores, your organization can benefit in several ways from using the multiplexor. The indirect connection between users and their message stores, coupled with the ease of reconfiguration of user accounts among messaging servers allows for the following benefits:

- **Simplified User Management**
  Because all users connect to one server (or more, if you have separate multiplexor machines for POP, IMAP, SMTP, or web access), you can preconfigure email clients and distribute uniform login information to all users. This simplifies your administrative tasks and reduces the possibility of distributing erroneous login information.
  For especially high-load situations, you can run multiple multiplexor servers with identical configurations and manage connections to them by DNS round robin or by using a load-balancing system.
  Because the multiplexors use information stored in the LDAP directory to locate each user's Messaging Server, moving a user to a new server is simple for the system administrator and

transparent to the user. The administrator can move a user's mailbox from one Messaging Server to another, and then update the user's entry in the LDAP directory. The user's mail address, mailbox access, and other client preferences need not change.

- **Improved Performance**
  If a message store grows prohibitively large for a single machine, you can balance the load by moving some of the message store to another machine.
  You can assign different classes of users to different machines. For example, you can choose to locate premium users on a larger and more powerful machine.
  The multiplexors perform some buffering so that slow client connections (through a modem, for example) do not slow down the Messaging Server.
- **Decreased Cost**
  Because you can efficiently manage multiple Messaging Server hosts with a multiplexor, you might be able to decrease overall costs by purchasing several small server machines that together cost less than one very large machine.
- **Better Scalability**
  With the multiplexors, your configuration can expand easily. You can incrementally add machines as your performance or storage-capacity needs grow, without replacing your existing investment.
- **Minimum User Downtime.** Using the multiplexors to spread a large user base over many small store machines isolates user downtime. When an individual server fails, only its users are affected.
- **Increased Security**
  You can use the server machine on which the multiplexor is installed as a firewall machine. By routing all client connections through this machine, you can restrict access to the internal message store machines by outside computers. The multiplexors support both unencrypted and encrypted communications with clients.

About Messaging Multiplexor

The Messaging Multiplexor (MMP) is a specialized messaging server that acts as a single point of connection to multiple back-end messaging servers. With Messaging Multiplexor, large-scale messaging service providers can distribute POP and IMAP user mailboxes across many machines to increase message store capacity. All users connect to the single multiplexor server, which redirects each connection to the appropriate messaging server.

If you provide electronic mail service to many users, you can install and configure the Messaging Multiplexor so that an entire array of Messaging Server hosts appear to your mail users to be a single host.

The Messaging Multiplexor is provided as part of Messaging Server. You can install the MMP at the same time you install Messaging Server or other Communications Suite servers, or you can install the MMP separately at a later time. The MMP supports the following items:

- Both unencrypted and encrypted (SSL) communications with mail clients
- Client certificate-based authentication, described in Certificate-Based Client Authentication
- User pre-authentication, described in User Pre-Authentication
- Virtual domains that listen on different IP addresses and automatically append domain names to user IDs, described in MMP Virtual Domains
- Multiple installations of the MMP on different servers
- Enhanced LDAP searching
- POP before SMTP service for legacy POP clients, described in Enabling POP Before SMTP.

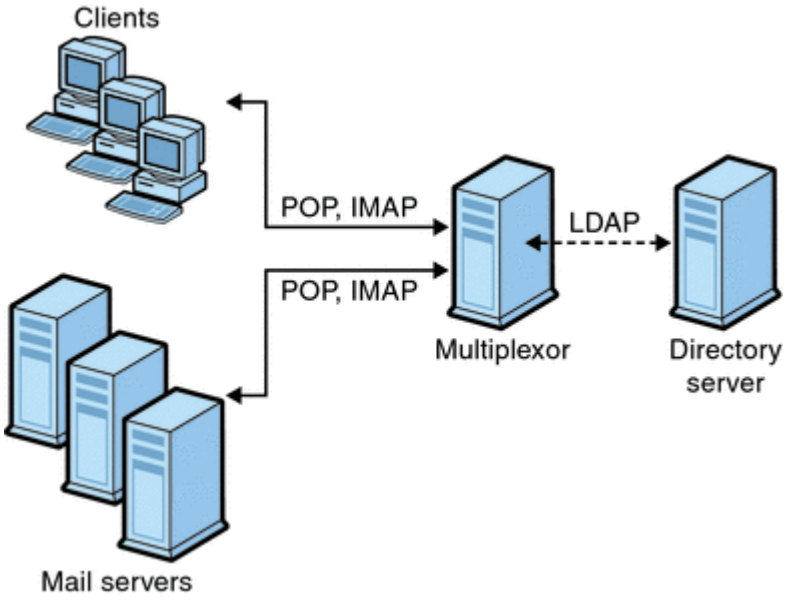
This section consists of the following subsections:

- MMP Virtual Domains
- About SMTP Proxy

How the Messaging Multiplexor Works

The MMP is a multithreaded server that facilitates distributing mail users across multiple server machines. The MMP handles incoming client connections destined for other server machines (the machines on which user mailboxes reside). Clients connect to the MMP itself, which determines the correct server for the users, connects to that server, and then passes data between the client and server. This capability allows Internet service providers and other large installations to spread message stores across multiple machines (to increase capacity) while providing the appearance of a single mail host for users (to increase efficiency) and for external clients (to increase security). The following figure shows clients and servers in an MMP installation.

Clients and Servers in an MMP Installation



All POP, IMAP, and SMTP clients work with the Messaging Multiplexor. The MMP accepts connections, performs LDAP directory lookups, and routes the connections appropriately. As is typical with other mail server installations, each user is assigned a specific address and mailbox on a specific Messaging Server. However, all connections are routed through the MMP.

In more detail, these are the steps involved in establishing a user connection:

1. A user's client connects to the MMP, which accepts preliminary authentication information (user name).
2. The MMP queries the Directory Server to determine which Messaging Server contains that user's mailbox.
3. The MMP connects to the proper Messaging Server, replays authentication, then acts as a pass-through pipe for the duration of the connection.

Encryption (SSL) Option

Messaging Multiplexor supports both unencrypted and encrypted (SSL) communications between the Messaging Server(s) and their mail clients. The current version of Messaging Server supports the new certificate database format (`cert8.db`).

When SSL is enabled, the MMP IMAP supports both STARTTLS on the standard IMAP port and IMAP+SSL on port 993. The MMP can also be configured to listen on port 995 for POP+SSL.

To enable SSL encryption for your IMAP, POP, and SMTP services, edit the `ImapProxyAService.cfg`, `PopProxyAService.cfg`, and `SmtpProxyAService.cfg` files, respectively. You must also edit the `default:ServiceList` option in the `AService.cfg` file to include the list of all IMAP, POP, and SMTP server ports regardless of whether or not they are secure. See Configuring MMP with SSL for details.

By default, SSL is not enabled since the SSL configuration parameters are commented out. To enable SSL, you must install an SSL server certificate. Then, you should uncomment and set the SSL parameters. For a list of the SSL parameters, see the `ssl*` parameters in the MMP Reference.

Certificate-Based Client Authentication

The MMP can use a certificate mapping file (`certmap.conf`) to match a client's certificate to the correct user in the Users/Groups Directory Server.

To use certificate-based client authentication, you must also enable SSL encryption as described in Encryption (SSL) Option.

You also have to configure a store administrator. You can use the mail administrator, but it is recommended that you create a unique user ID, such as `mmpstore` for this purpose so that you can set permissions as needed.

The MMP does not support `certmap` plug-ins. Instead, the MMP accepts enhanced `DNComps` and `FilterComps` property value entries in the `certmap.conf` file. These enhanced format entries use the form:

```
<mapname>:DNComps <FROMATTR=TOATTRmapname>:FilterComps <FROMATTR=TOATTR>
```

So that a *FROMATTR* value in a certificate's subjectDN can be used to form an LDAP query with the *TOATTR=value* element. For example, a certificate with a subjectDN of "`cn=Pilar Lorca, ou=pilar, o=siroe.com`" could be mapped to an LDAP query of "`(uid=pilar)`" with the line:

```
<mapname>:FilterComps ou=uid
```

To Enable Certificate-based Authentication for Your IMAP or POP Service

1. Decide on the user ID you intend to use as store administrator.
   While you can use the mail administrator for this purpose, it is recommended that you create a unique user ID for store administrator (for example, `mmpstore`).
2. Make sure that SSL encryption is (or will be) enabled as described in Encryption (SSL) Option.
3. Configure the MMP to use certificate-based client authentication by specifying the location of the `certmap.conf` file in your configuration files.
4. Install at least one trusted CA certificate, as described in Configuring Encryption and Certificate-Based Authentication in Unified Configuration.

User Pre-Authentication

The MMP provides you with the option of pre-authenticating users by binding to the directory as the incoming user and logging the result.

> **Note**
> Enabling user pre-authentication reduces server performance.

The log entries are in the format:

date time `(sid 0x`*hex*`)` user *name* `pre-authenticated - client IP` *address,* `server IP` *address*

Where *date* is in the format `yyyymmdd`, *time* is in the time configured on the server in the format `hhmmss`, *hex* is the session identifier (`sid`) represented as a hexidecimal number, the `user` *name* includes the virtual domain (if any), and the IP address is in dot-quad format.

MMP Virtual Domains

An MMP *virtual domain* is a set of configuration settings associated with one or more server IP addresses. The primary use of this feature is to provide different default domains for each server IP address.

A user can authenticate to the MMP with either a short-form userID or a fully qualified userID in the form `user@domain`. When a short-form userID is supplied, the MMP will append the `DefaultDomain` setting, if specified. Consequently, a site which supports multiple hosted domains can permit the use of short-form user IDs simply by associating a server IP address and MMP virtual domain with each hosted domain.

The recommended method for locating the user subtree for a given hosted domain is by using the `inetDomainBaseDN` attribute in the LDAP domain tree entry for that domain. The MMP's `LdapUrl` setting is not suitable for this purpose since the back-end mail store servers also need to look up the user in LDAP and do not support virtual domains.

When Oracle LDAP Schema 2 is enabled (see *Unified Communications Suite Schema Reference*), the user subtree for the specified domain is all the users in the subtree below the organization node for that domain.

To enable virtual domains, edit the `ImapProxyAService.cfg`, `PopProxyAService.cfg`, or `SmtpProxyAService.cfg` files in the instance directory such that the `VirtualDomainFile` setting specifies the full path to the virtual domain mapping file.

Each entry of a virtual domain mapping file has the following syntax:

`vdmap` *name IP-address*
name:*parameter value*
name:*parameter value*
.
.
.

where:

`vdmap`	Is the keyword to introduce the virtual domain mapping block
name	Is a name you choose to associate the IP address with the configuration parameters
IP-address	Is in dot-quad format
parameter and *value* pairs	Are the configuration parameters specific to the virtual domain

When set, virtual domain configuration parameter values override global configuration parameter values.

Additional IP addresses can be associated with a virtual domain by using the `IP` parameter, for example:

```
vdmap test 10.30.16.28
test:IP 10.30.16.29 10.30.16.30
test:DefaultDomain test.com
```

You can specify the following configuration parameters for a virtual domain:

```
AuthenticationLdapAttributes
AuthService
BindDN
BindPass
CertMap
ClientLookup
CRAMs
DefaultDomain
DomainDelim
DomainSearchFormat
HostedDomains
MailHostAttrs
PreAuth
ReplayFormat
RestrictPlainPasswords
SSLCertNicknames
SSLAdjustCipherSuites
StoreAdmin
StoreAdminPass
SearchFormat
TCPAccess
TCPAccessAttr
```

For detailed descriptions of these configuration parameters, see *Messaging Server Administration Reference* or the MMP Reference.

About SMTP Proxy

The MMP includes an SMTP proxy which is disabled by default. Most sites do not need the SMTP proxy because Internet Mail standards already provide an adequate mechanism for horizontal scalability of SMTP (DNS MX records).

The SMTP proxy is useful for the security features it provides. First, the SMTP proxy is integrated with the POP proxy to implement the POP before SMTP authorization facility required by some legacy POP clients. For more information, see the topic on designing a Messaging Server topology in *Unified Communications Suite Deployment Planning Guide*, and Enabling POP Before SMTP. In addition, an investment in SSL acceleration hardware can be maximized by using the SMTP proxy.

Setting Up the Messaging Multiplexor

During the initial runtime configuration of Messaging Server, you determined if you wanted to configure the MMP on a machine. You could either set it up on the same machine as your Messaging Server or set it up on a separate machine.

> **Note**
> MMP does not cache DNS results. A high quality caching DNS server on the local network is a requirement for a production deployment of Messaging Server.

The following sections describe how to set up the MMP:

More information about the MMP can be found in the following:

- Messaging Multiplexor Configuration

Before You Configure MMP

Before configuring the MMP:

1. Choose the machine on which you will configure the MMP. It is best to use a dedicated machine for the MMP.

   > **Note**
   > It is recommended that the MMP not be enabled on a machine that is also running either the POP or IMAP servers.
   > If you install MMP on the same machine as Messaging Server, you must make sure that the POP and IMAP servers are set to non-standard ports. That way, the MMP and Messaging Server ports will not conflict with one another.

2. On the machine that the MMP is to be configured, create a UNIX system user to be used by the MMP. This new user must belong to a UNIX system group. See the topic on creating UNIX system users and groups in *Messaging Server 8.0 Installation and Configuration Guide*.
3. Set up the Directory Server and its host machine for use with Messaging Server, if they are not already set up. See the topic on how to prepare Directory Server for Messaging Server configuration in *Messaging Server 8.0 Installation and Configuration Guide*.
4. If the MMP is upgraded before the backend servers, customers should set the `Capability` option in `ImapProxyAService.cfg` to match the response to the `capability` command from the older backend. This would be:
   `"IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS CHILDREN LANGUAGE XSENDER X-NETSCAPE XSERVERINFO"`
   Note that line break is for editorial clarity, and that the configuration value must be on one line and contained within quotes.

Multiplexor Configuration

To configure the MMP, you must use the Messaging Server configure program, which gives you the option of enabling the Messaging Multiplexor. For detailed information about the configure program, see the topic on creating the initial Messaging Server runtime configuration in *Messaging Server 8.0 Installation and Configuration Guide*.

To Configure the MMP

1. Install Messaging Server software on the machine where you are installing and configuring the MMP.
2. Configure the MMP by creating the Messaging Server Initial Runtime Configuration. For more information, see the topic on creating the initial Messaging Server runtime configuration in *Messaging Server 8.0 Installation and Configuration Guide*.

Multiplexor Files

The Messaging Multiplexor files are stored in the *msg-svr-base*`/config` configuration file directory. You must manually edit the configuration parameters in the Messaging Multiplexor configuration files listed in the following table. For a complete description of all MMP configuration parameters, see the topic on multiplexor configuration parameters in *Messaging Server Administration Reference*.

Messaging Multiplexor Configuration Files

File	Description
`PopProxyAService.cfg`	Configuration file specifying configuration variables used for POP services.
`PopProxyAService-def.cfg`	POP services configuration template. File comes into existence only after initial MMP start with `start-msg mmp`
`ImapProxyAService.cfg`	Configuration file specifying configuration variables used for IMAP services.
`ImapProxyAService-def.cfg`	IMAP services configuration template. File comes into existence only after initial MMP start with `start-msg mmp`
`AService.cfg`	Configuration file specifying which services to start and a few options shared by both POP and IMAP services.
`AService-def.cfg`	Configuration template specifying which services to start and a few options shared by both POP and IMAP services. File comes into existence only after initial MMP start with `start-msg mmp`.
`SmtpProxyAService.cfg`	Optional configuration file specifying configuration variables used for SMTP Proxy services. Required if you enable POP before SMTP; useful for maximizing support for SSL hardware even if POP before SMTP is not enabled. For more information on POP before SMTP, see Enabling POP Before SMTP.
`SmtpProxyAService-def.cfg`	Configuration template specifying configuration variables used for SMTP Proxy services. File comes into existence only after initial MMP start with `start-msg mmp`.

As an example, the `LogDir` and `LogLevel` parameters can be found in all configuration files. In `ImapProxyAService.cfg`, they are used to specify logging parameters for IMAP-related events; similarly, these parameters in `PopProxyAService.cfg` are used to configure logging parameters for POP-related events. In `SmtpProxyAService.cfg,` they are used to specify logging for SMTP Proxy-related events.

In `AService.cfg`, however, `LogDir` and `LogLevel` are used for logging MMP-wide failures, such as the failure to start a POP, IMAP, or SMTP service.

For descriptions of these parameters, see MMP Reference.

Note
When the MMP is configured or upgraded, the configuration template files will be overwritten.

Multiplexor Configuration Parameters

You control how the MMP operates by specifying various configuration parameters in the MMP configuration files.

See MMP Reference for the parameters you can set.

Note
To enable configuration parameters for different instances to be specified in the same configuration file, all the parameters are preceded with "default:" to indicate the default section. See the `ServiceList` parameter for more information.

Starting the Multiplexor

To start, stop, or refresh an instance of the Messaging Multiplexor, use the one of the commands in the following table. These commands are located in the *msg-svr-base*`/sbin` directory:

MMP Commands

Option	Description
`start-msg mmp`	Starts the MMP (even if one is already running).
`stop-msg mmp`	Stops the most recently started MMP.
`refresh mmp`	Causes an MMP that is already running to refresh its configuration without disrupting any active connections.

Modifying an Existing MMP

To modify an existing instance of the MMP, edit the `ImapProxyAService.cfg` and/or `PopProxyAService.cfg` configuration files as necessary. These configuration files are located in the *msg-svr-base*`/config` subdirectory.

Configuring MMP with SSL

To configure the MMP to use SSL, do the following:

Note
It is assumed that the MMP is installed on a machine that does not have a Message Store or MTA.

To Configure MMP with SSL

1. Install an SSL server certificate (see Configuring Encryption and Certificate-Based Authentication in Unified Configuration).
2. Edit the `ImapProxyAService.cfg` file and uncomment the relevant SSL settings (at least `SSLEnable` and `SSLPorts`). Additionally, you must edit the `AService.cfg` file and add `|993` after the `143` in the `ServiceList` setting if it's not already present.
3. If you want SSL and POP, edit the `PopProxyAService.cfg` file and uncomment the relevant SSL settings.
   Additionally, you must edit the `AService.cfg` file and add `|995` after the `110` in the `ServiceList` setting.
4. Make sure that the `BindDN` and `BindPass` options are set in the `ImapProxyAService.cfg` and `PopProxyAService.cfg` files.
   You should also set the `DefaultDomain` option to your default domain (the domain to use for unqualified user names).
   If you just want server-side SSL support, you are finished. Start the MMP with the following command in the *msg-svr-base*`/sbin` directory:
   `start-msg mmp`
5. If you do not want to use SSL between the MMP and the backend server, then set the `SSLBacksidePort` option to 0 in the `ImapProxyAService.cfg` or `PopProxyAService.cfg` MMP configuration files.

To Configure MMP with Client Certificate-based Login

If you want client certificate based login, do the following:

1. Get a copy of a client certificate and the CA certificate which signed it.
2. Import the CA certificate as a Trusted Certificate Authority (see Obtaining and Managing Certificates).
3. Use the Store Administrator you created during your Messaging Server installation.
   For more information, see the Specifying Administrator Access to the Message Store in Unified Configuration.
4. Create a `certmap.conf` file for the MMP. For example:

   ```
   certmap default default default:DNComps default:FilterComps e=mail
   ```

   This means to search for a match with the e field in the certificate DN by looking at the mail attribute in the LDAP server.
5. Edit your `ImapProxyAService.cfg` file and do the following:
   a. Set `CertMapFile` to `certmap.conf`
   b. Set `StoreAdmin` and `StorePass` to values from Step 3.
   c. Set `UserGroupDN` to the root of your Users and Groups tree.
6. If you want client certificates with POP3, repeat Step 5 for the `PopProxyAService.cfg` file.
7. If the MMP is not already running, start it with the following command in the *msg-svr-base*`/sbin` directory:
   `start-msg mmp`
8. Import the client certificate into your client. In Netscape Communicator, click the padlock (Security) icon, then select Yours under Certificates, then select Import a Certificate... and follow the instructions.

   Note
   All your users have to perform this step if you want to use client certificates everywhere.

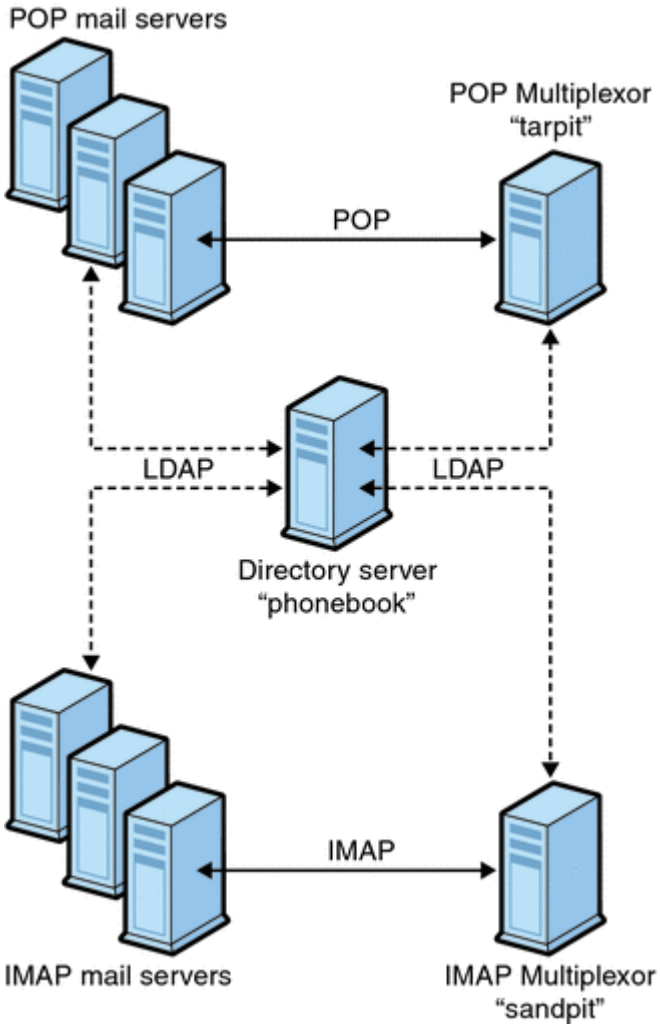
A Sample Topology

The fictional Siroe Corporation has two Messaging Multiplexors on separate machines, each supporting

several Messaging Servers. POP and IMAP user mailboxes are split across the Messaging Server machines, with each server dedicated exclusively to POP or exclusively to IMAP. (You can restrict client access to POP services alone by removing the `ImapProxyAService` entry from the `ServiceList` setting. Likewise, you can restrict client access to IMAP services alone by removing the `PopProxyAService` entry from the `ServiceList` setting.). Each Messaging Multiplexor also supports only POP or only IMAP. The LDAP directory service is on a separate, dedicated machine.

The following figure illustrates this topology.

Multiple MMPs Supporting Multiple Messaging Servers



IMAP Configuration Example

The IMAP Messaging Multiplexor in the Multiple MMPs Supporting Multiple Messaging Servers figure is installed on `sandpit`, a machine with two processors. This Messaging Multiplexor is listening to the standard port for IMAP connections (143). Messaging Multiplexor communicates with the LDAP server on the host `phonebook` for user mailbox information, and it routes the connection to the appropriate IMAP server. It overrides the IMAP capability string, provides a virtual domain file, and supports SSL communications.

This is its `ImapProxyAService.cfg` configuration file (Messaging Server 6):

```
default:LdapUrl ldap://phonebook.siroe.com/o=internet
default:LogDir /opt/SUNWmsgsr/config/log
default:LogLevel 5
default:BindDN "cn=Directory Manager"
default:BindPass secret
default:BacksidePort 143
default:Timeout 1800
default:Capability "IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE
UIDPLUS CHILDREN BINARY LANGUAGE XSENDER X-NETSCAPE XSERVERINFO"
default:SearchFormat (uid=%s)
default:SSLEnable yes
default:SSLPorts 993
default:SSLSecmodFile /opt/SUNWmsgsr/config/secmod.db
default:SSLCertFile /opt/SUNWmsgsr/config/cert8.db
default:SSLKeyFile /opt/SUNWmsgsr/config/key3.db
default:SSLKeyPasswdFile /opt/SUNWmsgsr/config/sslpassword.conf
default:SSLCipherSpecs all
default:SSLCertNicknames Siroe.com Server-Cert
default:SSLCacheDir /opt/SUNWmsgsr/config
default:SSLBacksidePort 993
default:VirtualDomainFile /opt/SUNWmsgsr/config/vdmap.cfg
default:VirtualDomainDelim @
default:ServerDownAlert "your IMAP server appears to be temporarily
out of service"
default:MailHostAttrs mailHost
default:PreAuth no
default:CRAMs no
default:AuthCacheSize 10000
default:AuthCacheTTL 900
default:AuthService no
default:AuthServiceTTL 0
default:BGMax 10000
default:BGPenalty 2
default:BGMaxBadness 60
default:BGDecay 900
default:BGLinear no
default:BGExcluded /opt/SUNWmsgsr/config/bgexcl.cfg
default:ConnLimits 0.0.0.0|0.0.0.0:20
default:LdapCacheSize 10000
default:LdapCacheTTL 900
default:HostedDomains yes
default:DefaultDomain Siroe.com
```

POP Configuration Example

The POP Messaging Multiplexor example in A Sample Topology is installed on `tarpit`, a machine with four processors. This Messaging Multiplexor is listening to the standard port for POP connections (110). Messaging Multiplexor communicates with the LDAP server on the host `phonebook` for user mailbox information, and it routes the connection to the appropriate POP server.

This is its `PopProxyAService.cfg` configuration file (Messaging Server 6):

```
default:LdapUrl ldap://phonebook.siroe.com/o=internet
default:LogDir /opt/SUNWmsgsr/config/log
default:LogLevel 5
default:BindDN "cn=Directory Manager"
default:BindPass password
default:BacksidePort 110
default:Timeout 1800
default:SearchFormat (uid=%s)
default:SSLEnable no
default:VirtualDomainFile /opt/SUNWmsgsr/config/vdmap.cfg
default:VirtualDomainDelim @
default:MailHostAttrs mailHost
default:PreAuth no
default:CRAMs no
default:AuthCacheSize 10000
default:AuthCacheTTL 900
default:AuthService no
default:AuthServiceTTL 0
default:BGMax 10000
default:BGPenalty 2
default:BGMaxBadness 60
default:BGDecay 900
default:BGLinear no
default:BGExcluded /opt/SUNWmsgsr/config/bgexcl.cfg
default:ConnLimits 0.0.0.0|0.0.0.0:20
default:LdapCacheSize 10000
default:LdapCacheTTL 900
default:HostedDomains yes
default:DefaultDomain Siroe.com
```

MMP Tasks

This section describes miscellaneous MMP configuration tasks. These include:

- To Configure Mail Access with MMP
- To Set a Failover MMP LDAP Server

To Configure Mail Access with MMP

The MMP does not make use of the `PORT_ACCESS` mapping table. If you wish to reject SMTP connections from certain IP addresses and you are using the MMP, you must use the `TCPAccess` option. The syntax of this option is the same as `mailDomainAllowedServiceAccess` (see *Messaging Server Administration Reference*). It is also described at Filter Syntax.

To Set a Failover MMP LDAP Server

It is possible to specify more than one LDAP server for the MMP so that if one fails another takes over. Modify your `PopProxyAservice.cfg` or `ImapProxyAservice.cfg` to the following:

`default:LdapUrl "ldap://`*ldap01.yourdomain ldap02.yourdomain*`/o=internet"`

> **Note**
> Make sure there is a space between the host names in the above configuration.

Chapter 11. Configuring Rewrite Rules

Configuring Rewrite Rules

This information describes how to configure rewrite rules in the `imta.cnf` file. Make sure you have read About MTA Services and Configuration before using this information.

Topics:

Messaging Server's address rewriting facility is the primary facility for manipulating and changing the host or domain portion of addresses. Messaging Server provides other facilities for address manipulation, such as aliases, the address reversal database, and specialized mapping tables. For best performance, however, rewrite rules should be used whenever possible to perform address manipulations.

Before You Begin

When you make changes to rewrite rules in the `imta.cnf` file, you must restart any programs or channels that load the configuration data only once when they start up, for example, the SMTP server, by using the `imsimta restart` command. If you are using a compiled configuration, you must recompile the configuration by running `imsimta cnbuild` and then restart.

For more information about compiling configuration information and starting programs, see *Messaging Server Administration Reference*.

Rewrite Rule Structure

Rewrite rules appear in the upper-half of the MTA configuration file, `imta.cnf`. 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. The example below shows the rewrite rule section of a partial configuration file.

```
! test.cnf - An example configuration file.
!
! This is only an example of a configuration file. It serves
! no useful purpose and should not be used in a real system.
!
a.com   $U@a-host
b.org   $U@b-host
c.edu   $U%c@b-daemon
d.com   $U%d@a-daemon

! Begin channel definitions
```

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 structure for rewrite rules is as follows:

```
pattern    template
pattern    template
pattern    template
```

pattern

Indicates the string to search for in the domain name. In Extracted Addresses and Host Names, the patterns are `a.com`, `b.org`, `c.edu`, and `d.com`.

If the pattern matches the domain part of the address, the rewrite rule is applied to the address. A blank space must separate the pattern from the template. For more information about pattern syntax, see Rewrite Rule Patterns and Tags.

template

is one of the following:

```
UserTemplate%DomainTemplate@ChannelTag[controls]
UserTemplate@ChannelTag[controls]
UserTemplate%DomainTemplate[controls]
UserTemplate@DomainTemplate@ChannelTag[controls]
UserTemplate@DomainTemplate@SourceRoute@ChannelTag[controls]
```

where:

UserTemplate specifies how the user part of the address is rewritten. Substitution sequences can be used to represent parts of the original address or the results of a database lookup. The substitution sequences are replaced with what they represent to construct the rewritten address. In Summary of Rewrite Rule Template Substitutions and Control Sequences, the `$U` substitution sequence is used. For more information, see Template Substitutions and Rewrite Rule Control Sequences.

DomainTemplate specifies how the domain part of the address is rewritten. Like the *UserTemplate*, the *DomainTemplate* can contain substitution sequences.

ChannelTag indicates the channel to which this message is sent. (All channel definitions must include a channel tag as well as a channel name. The channel tag typically appears in rewrite rules, as well as in its channel definition.)

controls limits the applicability of a rule. Some control sequences must appear at the beginning of the rule. Other controls must appear at the end of the rule. For more information about controls, see Template Substitutions and Rewrite Rule Control Sequences.

For more information about template syntax, see Rewrite Rule Templates.

Rewrite Rule Patterns and Tags

This section consists of the following subsections:

- A Rule to Match Percent Hacks
- A Rule to Match Bang-Style (UUCP) Addresses
- A Rule to Match Any Address
- Tagged Rewrite Rule Sets

Most rewrite rule patterns consist either of a specific host name that will match only that host or of a subdomain pattern that will match any host or domain in the entire subdomain.

For example, the following rewrite rule pattern contains a specific host name that will match the specified host only:

```
host.siroe.com
```

The next rewrite rule pattern contains a subdomain pattern that will match any host or domain in the entire subdomain:

```
.siroe.com
```

This pattern will not, however, match the exact host name `siroe.com`. To match the exact host name `siroe.com`, a separate `siroe.com` pattern would be needed.

The MTA attempts to rewrite host and domain names starting from the specific host name and then incrementally generalizing the name to make it less specific. This means that a more specific rewrite rule pattern will be preferentially used over more general rewrite rule patterns. For example, assume the following rewrite rule patterns are present in the configuration file:

```
hosta.subnet.siroe.com
.subnet.siroe.com
.siroe.com
```

Based on the rewrite rule patterns, an address of `jdoe@hosta.subnet.siroe.com` matches the `hosta.subnet.siroe.com` rewrite rule pattern. An address of `jdoe@hostb.subnet.siroe.com` matches the `.subnet.siroe.com` rewrite rule pattern. And an address of `jdoe@hostc.siroe.com` matches the `.siroe.com` rewrite rule pattern.

In particular, the use of rewrite rules incorporating subdomain rewrite rule patterns is common for sites on the Internet. Such a site will typically have a number of rewrite rules for its own internal hosts and subnets, and then include rewrite rules for the top-level Internet domains into its configuration from the file `internet.rules` (*msg-svr-base*`/config/internet.rules`).

To ensure that messages to Internet destinations (other than to the internal host destinations handled by using more specific rewrite rules) are properly rewritten and routed to an outgoing TCP/IP channel, ensure that the `imta.cnf` file contains the following information:

- Rewrite rules with patterns that match the top level Internet domains
- Templates that rewrite addresses matching such patterns to an outgoing TCP/IP channel

```
!    Ascension Island
 .AC                      $U%$H$D@TCP-DAEMON

[text removed for brevity]

!      Zimbabwe
 .ZW                      $U%$H$D@TCP-DAEMON
```

IP domain literals follow a similar hierarchical matching pattern, though with right-to-left (rather than left-to-right) matching. For example, the following pattern matches only and exactly the IP literal `[1.2.3.4]`:

```
[1.2.3.4]
```

The next pattern matches anything in the `1.2.3.0` subnet:

```
[1.2.3.]
```

In addition to the more common sorts of host or subdomain rewrite rule patterns already discussed, rewrite rules may also make use of several special patterns, summarized in Summary of Special Patterns for Rewrite Rules, and discussed in the following subsections.

Summary of Special Patterns for Rewrite Rules

Pattern	Description/Usage
`$*`	Matches any address. This rule, if specified, is tried first regardless of its position in the file.
$%	Percent Hack Rule. Matches any host/domain specification of the form A%B
`$!`	Bang-style Rule. Matches any host/domain specification of the form B!A
`[ ]`	IP literal match-all rule. Matches any IP domain literal.
`.`	Matches any host/domain specification. For example, `joe@[129.165.12.11]`

In addition to these special patterns, Messaging Server 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 rewrites, distinctions must be made in subsequent rewrites by controlling which rewrite rules match the address. For more information, see Tagged Rewrite Rule Sets.

A Rule to Match Percent Hacks

If the MTA 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`. (For more information about these address forms, see Rewrite Rule Templates.) 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 the MTA 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 later).

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) matches 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**
> Regarding substitution sequences, 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` in a match-all rule template! is of limited use.

Tagged Rewrite Rule Sets

As the rewrite process proceeds it might 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 later).

Tags are somewhat sticky. Once set, they 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, which is 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.

Rewrite Rule Templates

The following sections describe in more detail template formats for rewrite rules. The following table summarizes the template formats.

Summary of Template Formats for Rewrite Rules

Template	Usage
A%B	A becomes the new user/mailbox name, B becomes the new host/domain specification, rewrite again. See Repeated Rewrites Template, A%B.
A@B	Treated as A%B@B. See Ordinary Rewriting Templates, A%B@C or A@B.
A%B@C	A becomes the new user/mailbox name, B becomes the new host/domain specification, route to the channel associated with the host C. See Ordinary Rewriting Templates, A%B@C or A@B.
A@B@C	Treated as A@B@C@C. See Specified Route Rewriting Templates, A@B@C@D or A@B@C.
A@B@C@D	A becomes the new user/mailbox name, B becomes the new host/domain specification, insert C as a source route, route to the channel associated with the host D. See Specified Route Rewriting Templates, A@B@C@D or A@B@C.

Ordinary Rewriting Templates, A%B@C or A@B

The following template is the most common form of template. The rule is applied to the user part of the address and to the domain part of the address. The new address is then used to route the message to a specific channel (indicated by *ChannelTag*).

```
UserTemplate%DomainTemplate@ChannelTag[controls]
```

The next form of template is identical in application to the most common form of template. However, this form of template is possible only if *DomainTemplate* and *ChannelTag* are identical.

```
UserTemplate@ChannelTag[controls]
```

Repeated Rewrites Template, A%B

The following template format is used for meta-rules that require additional rewriting after application of the rule. After the rule is applied, the entire rewriting process is repeated on the resulting new address. (All other rewrite rule formats cause the rewriting process to terminate after the rule has been applied).

```
UserTemplate%DomainTemplate[controls]
```

While the special A%B form does cause rewriting of the current domain to restart, it is actually just a continuation of the current rewriting process. It does not rewrite the entire process from the beginning. It does not perform the `$*` pattern when it goes through the second time.

For example, the following rule has the effect of removing all occurrences of the `.removable` domain from the ends of addresses:

```
.removable $U%$H
```

Extreme care must be taken when using these repeating rules as careless use can create a "rules loop". For this reason meta-rules should only be used when absolutely necessary. Be sure to test meta-rules with the `imsimta test -rewrite` command. For more information on the `test -rewrite` command, see *Messaging Server Administration Reference*.

Specified Route Rewriting Templates, A@B@C@D or A@B@C

The following template format works in the same way as the more common template `UserTemplate%DomainTemplate@ChannelTag` (note the difference in the first separator character), except that *ChannelTag* is inserted into the address as a source route. The message is then routed to *ChannelTag*:

```
UserTemplate@DomainTemplate@Source-Route@ChannelTag[controls]
```

The rewritten address becomes `@route:user@domain`. The following template is also valid:

```
UserTemplate@DomainTemplate@ChannelTag[controls]
```

For example, the following rule rewrites the address `jdoe@com1` into the source-routed address `@siroe.com:jdoe@com1`. The channel tag becomes `siroe.com`:

```
com1  $U@com1@siroe.com
```

Case Sensitivity in Rewrite Rule Templates

Unlike the patterns in rewrite rules, character case in templates is preserved. This is necessary when using rewrite rules to provide an interface to a mail system that is sensitive to character case. Note that substitution sequences like $U and $D that substitute material extracted from addresses also preserve the original case of characters.

When it is desirable to force substituted material to use a particular case, for example, to force mailboxes to lowercase on UNIX systems, special substitution sequences can be used in templates to force substituted material to the desired case. Specifically, $\ forces subsequent substituted material into lower case, $^ forces subsequent substituted material into upper case, and $_ says to use the original case.

For example, you can use the following rule to force mailboxes to lowercase for `unix.siroe.com` addresses:

```
unix.siroe.com    $\$U$_%unix.siroe.com
```

How the MTA Applies Rewrite Rules to an Address

The following steps describe how the MTA applies rewrite rules to a given address:

1. The MTA extracts the first host or domain specification from an address.
   An address can specify more than one host or domain name as in the case: `jdoe%hostname@siroe.com.`
2. After identifying the first host or domain name, the MTA conducts a search that scans for a rewrite rule whose pattern matches the host or domain name.
3. When the matching rewrite rule is found, the MTA rewrites the address according to the template portion of that rule.
4. Finally, the MTA compares the channel tag with the host names that are associated with each channel.
   If a match is found, the MTA enqueues the message to the associated channel; otherwise, the rewrite process fails. If the matching channel is the local channel, some additional rewriting of the address may take place by looking up the alias database and alias file.

These steps are described in more detail in the subsections that follow.

Note
Using a channel tag that does not belong to any existing channel will cause messages whose addresses match this rule to be bounced. That is, it makes the matching messages nonroutable.

This section consists of the following subsections:

Step 1. Extract 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:

1. Hosts in source routes (read from left to right)
2. Hosts appearing to the right of the "at" sign (@)
3. Hosts appearing to the right of the last single percent sign (%)
4. 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 the following table.

Extracted Addresses and Host Names

Address	First Host Domain Specification	Comments
user@a	a	A "short-form" domain name.
user@a.b.c	a.b.c	A "fully qualified" domain name (FQDN).
user@[0.1.2.3]	[0.1.2.3]	A "domain literal".
@a:user@b.c.d	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	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 the Messaging Server MTA.

The special interpretation of repeated percent signs is used to allow percent signs as part of local user names; this might be useful in handling some 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 the Messaging Server MTA.

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.

Step 2. Scan the Rewrite Rules

Once the first host or domain specification has been extracted from the address, the MTA 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. The MTA 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, non-asterisk part of `spec_2` is converted to an asterisk. For example, if `spec_2` is `a.b.c` then it is changed to `*.b.c`; if `spec_2` is `*.b.c`, then it is changed to `*.*.c`. The matching procedure is exited if a match is found.
- If no match is found then the first part, including any leading period, of the comparison string `spec_1` is removed. 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.siroe.edu` is to be rewritten. This causes the MTA to look for the following patterns in the given order:

```
sc.cs.siroe.edu
*.cs.siroe.edu
.cs.siroe.edu
*.*.siroe.edu
.siroe.edu
*.*.*.edu
.edu
*.*.*.*
.
```

Step 3. Rewrite Address According to Template

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.
3. A channel tag that identifies an existing MTA channel to which messages to this address should be sent.

Step 4. Finish the Rewrite Process

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

- If the channel tag is associated neither with the local channel nor a channel marked with the `routelocal` channel keyword, 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 channel tag matches the local channel or a channel marked `routelocal` 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, non-routelocal channel is found. This iterative mechanism is how the MTA provides support for source routing. In effect, superfluous routes through the local system and routelocal systems 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, the MTA 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 Rewrite

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 MTA with incorrect or illegal addresses.

Handling Domain Literals

Domain literals are handled specially during the rewriting process. If a domain literal appearing in the domain portion of an address does not match a rewrite rule pattern as is, 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 `"."`.

Template Substitutions and Rewrite Rule Control Sequences

Substitutions are used to rewrite 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.

This section consists of the following subsections:

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

```
$U@siroe.com
```

Control 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 example, 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. The following rewrite rule only applies to (rewrite) envelope To: addresses of the form `user@siroe.com`:

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

If a domain or host specification matches the pattern portion of a rewrite rule but doesn't meet all of the criteria imposed by a control sequences in the rule's template, then the rewrite rule fails and the rewriter continues to look for other applicable rules.

The following table summarizes the template substitutions and control sequences.

Summary of Rewrite Rule Template Substitutions and Control Sequences

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.

$*n*A	Inserts the nth left character of the current address starting from position 0. The entire address is inserted if n is omitted.
$*n*X	Inserts the nth left component of the mailhost starting from 0. The entire mailhost is inserted if n is omitted.
$0U	Local part (username) from original address, minus any subaddress.
$1U	Subaddress, if any, from local part (username) of 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.
$=	Forces subsequent substituted characters to undergo quoting appropriate for insertion into LDAP search filters.
$W	Substitutes in a random, unique string.
$]...[	LDAP search URL lookup.
$.	Establish a string which will be processed as the mapping entry result in the event of a temporary LDAP lookup failure.
$(text)	General database substitution; rule fails if lookup fails.
${...}	Applies specified mapping to supplied string.
$[...]	Invoke customer supplied routine; substitute in result.
$&*n*	The *nth* part of unmatched (or wildcarded) host, counting from left to right, starting from 0.
$!*n*	The *nth* part of unmatched (or wildcarded) 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.
$*n*D	Portion of domain specification that matched, preserving from the nth leftmost part starting from 0
$*n*H	Portion of host/domain specification that didn't match, preserving from the nth leftmost part starting from 0
Control Sequence	**Effect on Rewrite Rule**
$1M	Apply only if the channel is an internal reprocessing channel.
$1N	Apply only if the channel is not an internal reprocessing channel.
$1~	Perform any pending channel match checks. If the checks fail, successfully terminate processing of the current rewrite rule template.
$A	Apply if host is to the right of the at sign

$B	Apply only to header/body addresses
$C*channel*	Fail if sending to *channel*
$E	Apply only to envelope addresses
$F	Apply only to forward-directed (e.g., To\:) addresses
$M*channel*	Apply only if *channel* is rewriting the address
$N*channel*	Fail if *channel* is rewriting the address
$P	Apply if host is to the right of a percent sign
$Q*channel*	Apply if sending to *channel*
$R	Apply only to backwards-directed (e.g., From\:) addresses
$S	Apply if host is from a source route
$T*newtag*	Set the rewrite rule tag to newtag
$V*host*	Fail if the host name is not defined in the LDAP directory (either in the DC tree or as a virtual domain). If the LDAP search times out, the remainder of the rewrite pattern from directly after the character following the host name is replaced with the MTA option string DOMAIN_FAILURE.
$X	Apply if host is to the left of an exclamation point
$Z*host*	Fail if the host name is defined in the LDAP directory (either in the DC tree or as a virtual domain). If the LDAP search times out, the remainder of the rewrite pattern from directly after the character following the host name is replaced with the MTA option string DOMAIN_FAILURE.
$*n*T	Overrides the default ALIAS_MAGIC setting, where *n* is an appropriate value for the ALIAS_MAGIC MTA option. Overrides the setting for the domain when the rule matches during alias expansion.
$?*errmsg*	If rewriting fails, return *errmsg* instead of the default error message. The error message must be in US ASCII.
$*number*?*errmsg*	If rewriting fails, return *errmsg* instead of the default error message, and set the SMTP extended error code to *a*.*b*.*c*: • *a* is *number*/ 1000000 (the first digit) • *b* is (*number*/1000) remainder 1000 (the value of the digits 2 through 4) • *c* is *number* remainder 1000 (the value of the last three digits. The following example sets the error code to 3.45.89: `$3045089?the snark is a boojum`

Username and Subaddress Substitution, $U, $0U, $1U

Any occurrences of $U in the template are replaced with the username (RFC 822 "local-part") from the original address. Note that user names of the form a."b" will be replaced by "a.b" as RFC2822 deprecates the former syntax from RFC 822 and it is expected that the latter usage will become mandatory in the future.

Any occurrences of $0U in the template are replaced with the username from the original address, minus any subaddress and subaddress indication character (+). Any occurrences of $1U in the template are

replaced with the subaddress and subaddress indication character, if any, from the original address. So note that $0U and $1U are complementary pieces of the username, with $0U$1U being equivalent to a simple $U.

Host/Domain and IP Literal Substitutions, $D, $H, $nD, $nH, $L

Any occurrences of $H are replaced with the portion of the host/domain specification that was not matched by the rule. Any occurrences of $D are replaced by the portion of the host/domain specification that was matched by the rewrite rule. The $nH and $nD characters are variants that preserve the normal $H or $D portion from the nth leftmost part starting counting from 0. That is, $nH and $nD omit the leftmost n parts (starting counting from 1) of what would normally be a $H or $D, substitution, respectively. In particular, $0H is equivalent to $H and $0D is equivalent to $D.

For example, assume the address `jdoe@host.siroe.com` matches the following rewrite rule:

```
host.siroe.com    $U%$1D@TCP-DAEMON
```

The resulting address is `jdoe@siroe.com` with `TCP-DAEMON` used as the outgoing channel. Here where $D would have substituted in the entire domain that matched, `host.siroe.com`, the $1D instead substitutes in the portions of the match starting from part 1 (part 1 being `siroe`), so substitutes in `siroe.com`.

$L substitutes the portion of a domain literal that was not matched by the rewrite rule.

Literal Character Substitutions, $$, $%, $@

The $, %, and @ characters are normally metacharacters in rewrite rule templates. To perform a literal insertion of such a character, quote it with a dollar character, $. That is, $$ 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).

LDAP Query URL Substitutions, $]...[

A substitution of the form `$]ldap-url[` is interpreted as an LDAP query URL and the result of the LDAP query is substituted. Standard LDAP URLs are used with the host and port omitted. The host and port are instead specified in the `msg.conf` file (`local.ldaphost` and `local.ldapport` attributes).

> Starting with **Messaging Server 7 Update 4**, you can specify the LDAP host and port in the URL, if desired. Prior to Messaging Server 7 Update 4, it was required that you omit the LDAP host and port, and you specified these values in `configutil` parameters. With the updated LDAP library delivered in Messaging Server 7 Update 4, if you specify the LDAP host and port in the URL, they override values specified in the `configutil` parameters.

That is, the LDAP URL should be specified as follows where the square bracket characters, [ ], indicate optional portions of the URL:

```
ldap:///dn[?attributes[?scope?filter]]
```

The `dn` is required and is a distinguished name specifying the search base. The optional attributes, scope, and filter portions of the URL further refine what information to return. For a rewrite rule, the

desired attributes to specify returning might be a `mailRoutingSystem` attribute (or some similar attribute). The scope may be any of base (the default), one, or sub. And the desired filter might be to request the return of the object whose `mailDomain` value matches the domain being rewritten.

If the LDAP directory schema includes attributes `mailRoutingSystem` and `mailDomain`, then a possible rewrite rule to determine to which system to route a given sort of address might appear as the following where here the LDAP URL substitution sequence $D is used to substitute in the current domain name into the LDAP query constructed:

```
.siroe.com \
  $U%$H$D@$]ldap:///o=siroe.com?mailRoutingSystem?sub? \
  (mailDomain=$D)
```

For ease in reading, the backslash character is used to continue the single logical rewrite rule line onto a second physical line. The following table lists the LDAP URL Substitution Sequences.

LDAP URL Substitution Sequences

Substitution Sequence	Description
$$	Literal $ character
$.	Establishes a string which will be processed as the mapping entry result in the event of a temporary LDAP lookup failure. By default a temporary failure string remains set only for the duration of the current rule. "$.." can be used to return to the default state where no temporary failure string is set and temporary LDAP failures cause mapping entry or rewrite rule failure. Note that all errors other than failure to match an entry in the directory are considered to be temporary errors; in general it isn't possible to distinguish between errors caused by incorrect LDAP URLs and errors caused by directory server configuration problems.
$~ *account*	Home directory of user account
$A	Address
$D	Domain name
$H	Host name (first portion of fully qualified domain name)
$L	Username minus any special leading characters such as ~ or _
$S	Subaddress
$U	Username

The MTA now caches URL results from lookups done in rewrite rules and mappings. This new URL result cache is controlled by two new MTA options, `URL_RESULT_CACHE_SIZE` (default 10000 entries) and `URL_RESULT_CACHE_TIMEOUT` (default 600 seconds).

General Database Substitutions, $(...)

A substitution of the form $(text) is handled specially. The text part is used as a key to access the special general text database. This database consists of the file specified with the `IMTA_GENERAL_DATABASE` option in the *msg-svr-base*`/config/imta_tailor` file, which is usually the file *msg-svr-base* `/db/generaldb.db`.

If "text-string" is found in the database, the corresponding template from the database is substituted. If "text-string" does not match an entry in the database, the rewrite process fails; it is as if the rewrite rule

never matched in the first place. If the substitution is successful, the template extracted from the database is re-scanned for additional substitutions. However, additional $(text) substitutions from the extracted template are prohibited in order to prevent endless recursive references.

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

```
.SIROENET $($H)
```

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

If a general text database exists it should be world readable to insure that it operates properly. See MTA Text Databases for more information.

Apply Specified Mapping, ${...}

A substitution of the form.SIROENET $($H) `${mapping,argument}` is used to find and apply a mapping from the MTA 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 MTA rewriting process to be extended in various complex ways. For example, the username part of an address can be selectively analyzed and modified, which normally isn't a feature the MTA rewriting process is capable of.

Customer-supplied Routine Substitutions, $[...]

A substitution of the form `$[`*image*`,`*routine*`,`*argument*`]` is used to find and call a customer-supplied routine. At run-time on UNIX, the MTA uses `dlopen` and `dlsym` to dynamically load and call the specified routine from the shared library image. The routine is then called as a function with the following argument list:

```
status := <routine> (<argument>, <arglength>, <result>, <reslength>)
```

argument and *result* are 252 byte long character string buffers. On UNIX, *argument* and *result* are passed as a pointer to a character string, (for example, in C, as `char*`.) *arglength* and *reslength* are signed, long integers passed by reference. On input, *argument* contains the argument string from the rewrite rule 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 rewrite rule template. The routine should return 0 if the rewrite rule should fail and -1 if the rewrite rule should succeed.

This mechanism allows the rewriting process to be extended in all sorts of complex ways. For example, a call to some type of name service could be performed and the result used to alter the address in some fashion. Directory service lookups for forward pointing addresses (e.g., To: addresses) to the host `siroe.com` might be performed as follows with the following rewrite rule. The $F, described in Direction-and-Location-Specific Rewrite Rules ($B, $E, $F, $R) causes this rule to be used only for forward pointing addresses:

```
siroe.com $F$[LOOKUP_IMAGE,LOOKUP,$U]
```

A forward pointing address `jdoe@siroe.com` will, when it matches this rewrite rule, cause `LOOKUP_IMAGE` (which is a shared library on UNIX) to be loaded into memory, and then cause the routine `LOOKUP` called with `jdoe` as the argument parameter. The routine `LOOKUP` might then return a different address, say, `John.Doe%eng.siroe.com` in the result parameter and the value -1 to indicate that the rewrite rule succeeded. The percent sign in the result string (see Repeated Rewrites Template, A%B} `John.Doe@eng.siroe.com` as the address to be rewritten.

On UNIX systems, the site-supplied shared library image should be world readable.

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 the following table.

Single Field Substitutions

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

Suppose the address `jdoe@eng.siroe.com` matches the following rewrite rule:

```
*.SIROE.COM     $U%$&0.siroe.com@mailhub.siroe.com
```

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

Unique String Substitutions

Each use of the $W control sequence inserts a text string composed of upper case letters and numbers that is designed to be unique and not repeatable. $W is useful in situation where non-repeating address information must be constructed.

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

It is possible to have rewrite rules that 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 MTA 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 MTA 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.

For example, `$M`*channel* causes the rule to fail if *channel* is not currently doing the rewriting. `$N`*channel* causes the rule to fail if *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)

It is possible to have rewrite rules whose application is dependent upon the channel to which a message is being enqueued. This is useful when there are two names for some host, 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.

For example, `$Q`*channel* causes the rule to fail if *channel* is not the destination. For another example, `$C` *channel* causes the rule to fail if *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 Rewrite Rules ($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 be applied if the address is forward pointing. The control sequence $R causes the rewrite to be applied if the address is reverse 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. Some situations might require specialized handling.

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

- $S specifies that the rule can match a host extracted from a source route.
- $A specifies that the rule can match a host found to the right of the @ sign.
- $P specifies that the rule can match a host found to the right of a % sign.
- $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 the 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.

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 at 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$Tmtcp-force|

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 MTA 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 will be returned as a result.

For example, assume the final rewrite rule in the MTA configuration file is as follows:

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

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

Handling Large Numbers of Rewrite Rules

The MTA always reads in all the rewrite rules from the `imta.cnf` 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 MTA 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, the MTA checks for the existence of the 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. The IMTA_DOMAIN_DATABASE attribute is stored in the `imta_tailor` file. The default location for the database is *msg-svr-base*`/data/db/domaindb.db`.

> **Note**
> Do not edit this file by hand.

Testing Rewrite Rules

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

```
imsimta test -rewrite -debug joe@siroe.com
```

For a detailed description of the `imsimta test -rewrite` utility, see *Messaging Server Administration Reference*.

Rewrite Rules Example

The following example provides sample rewrite rules and how sample addresses would be rewritten by the rules.

Suppose the configuration file for the system `SC.CS.SIROE.EDU` contained the rewrite rules shown in the following example:

```
sc                      $U@sc.cs.siroe.edu
sc1                     $U@sc1.cs.siroe.edu
sc2                     $U@sc2.cs.siroe.edu
*                       $U%$&amp;0.cs.siroe.edu
*.cs                    $U%$&amp;0.cs.siroe.edu
*.cs.siroe              $U%$&amp;0.cs.siroe.edu
*.cs.siroe.edu          $U%$&amp;0.cs.siroe.edu@ds.adm.siroe.edu
sc.cs.siroe.edu         $U@$D
sc1.cs.siroe.edu        $U@$D
sc2.cs.siroe.edu        $U@$D
sd.cs.siroe.edu         $U@sd.cs.siroe.edu
.siroe.edu              $U%$H.siroe.edu@cds.adm.siroe.edu
.edu                    $U@$H$D@gate.adm.siroe.edu
[]                      $U@[$L]@gate.adm.siroe.edu
```

The following table shows some sample addresses and how they would be rewritten and routed according to the rewrite rules.

Sample Addresses and Rewrites

Initial address	Rewritten as	Routed to
user@sc	user@sc.cs.siroe.edu	sc.cs.siroe.edu
user@sc1	user@sc1.cs.siroe.edu	sc1.cs.siroe.edu
user@sc2	user@sc2.cs.siroe.edu	sc2.cs.siroe.edu
user@sc.cs	user@sc.cs.siroe.edu	sc.cs.siroe.edu
user@sc1.cs	user@sc1.cs.siroe.edu	sc1.cs.siroe.edu
user@sc2.cs	user@sc2.cs.siroe.edu	sc2.cs.siroe.edu
user@sc.cs.siroe	user@sc.cs.siroe.edu	sc.cs.siroe.edu
user@sc1.cs.siroe	user@sc1.cs.siroe.edu	sc1.cs.siroe.edu
user@sc2.cs.siroe	user@sc2.cs.siroe.edu	sc2.cs.siroe.edu
user@sc.cs.siroe.edu	user@sc.cs.siroe.edu	sc.cs.siroe.edu
user@sc1.cs.siroe.edu	user@sc1.cs.siroe.edu	sc1.cs.siroe.edu
user@sc2.cs.siroe.edu	user@sc2.cs.siroe.edu	sc2.cs.siroe.edu
user@sd.cs.siroe.edu	user@sd.cs.siroe.edu	sd.cs.siroe.edu
user@aa.cs.siroe.edu	user@aa.cs.siroe.edu	ds.adm.siroe.edu
user@a.eng.siroe.edu	user@a.eng.siroe.edu	cds.adm.siroe.edu
user@a.cs.sesta.edu	user@a.cs.sesta.edu	gate.adm.siroe.edu – route inserted
user@b.cs.sesta.edu	user@b.cs.sesta.edu	gate.adm.siroe.edu – route inserted
user@[1.2.3.4]	user@[1.2.3.4]	gate.adm.siroe.edu – route inserted

Basically, what these rewrite rules say is: If the host name is one of our short-form names (`sc`, `sc1` or `sc2`) or if it is one of our full names (`sc.cs.siroe.edu`, and so on), expand it to our full name and route it to us. Append `cs.cmu.edu` to one part short-form names and try again. Convert one part followed by `.cs` to one part followed by `.cs.siroe.edu` and try again. Also convert `.cs.siroe` to `.cs.siroe.edu` and try again.

If the name is `sd.cs.siroe.edu` (some system we connect to directly, perhaps) rewrite and route it there. If the host name is anything else in the `.cs.siroe.edu` subdomain, route it to `ds.cs.siroe.edu` (the gateway for the `.cs.siroe.edu` subdomain). If the host name is anything else in the `.siroe.edu` subdomain route it to `cds.adm.siroe.edu` (the gateway for the `.siroe.edu` subdomain). If the host name is anything else in the `.edu` top-level domain route it to `gate.adm.siroe.edu` (which is presumably capable of routing the message to its proper destination). If a domain literal is used send it to `gate.adm.siroe.edu` as well.

Most applications of rewrite rules (like the previous example) will not change the username (or mailbox) part of the address in any way. The ability to change the username part of the address is used when the MTA is used to interface to mailers that do not conform to RFC 822 – mailers where it is necessary to stuff portions of the host/domain specification into the username part of the address. This capability should be used with great care if it is used at all.

Chapter 12. Configuring Channel Definitions

Configuring Channel Definitions

For information about configuring channel definitions, see Channel configuration.

Chapter 13. Creating and Managing Mailing Lists

Creating and Managing Mailing Lists in Oracle Communications Messaging Server

This information describes how to create mailing list entries in Messaging Server. As installed, mailing list entries are created in the Organization Tree. If your system is still using a SIMS DC Tree, then mailing list entries are created in the DC Tree. Attribute descriptions are overviews only. See *Unified Communications Suite Schema Reference* for complete attribute descriptions.

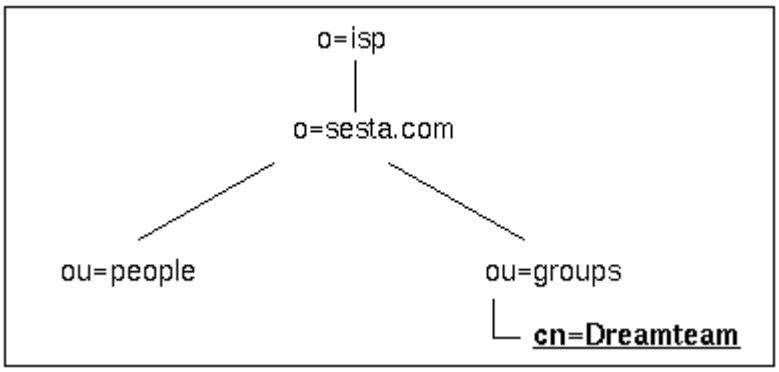
Topics:

Mailing Lists Overview

Mailing lists (as well as user and domain information) are stored as LDAP entries in the Directory Server. Creating and managing mailing lists consists of creating and modifying mailing list entries in the LDAP directory. To create and modify mailing list entries, you can use either the Delegated Administrator or modify LDAP directory information directly. In general, the Messaging Server documentation does not describe how to directly modify the LDAP directory. See the Directory Server documentation for more information on using LDAP tools.

The following figure shows where mailing list entries are created in the organization tree.

Example Organization Tree



Mailing list entries are created in the `ou=group` containers of the organization tree, as the preceding

figure shows. Following is an example LDIF record for a mailing list.

The following example shows an LDIF Record for a Mailing List:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: users 12/3/12
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
```

The following sections describe each of the object classes and attributes used in creating this mailing list:

- Mail List Attributes
- groupOfUniqueNames
- inetMailGroup
- inetLocalMailRecipient
- nsManagedMailList

Mail List Attributes

- `dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp`
  The distinguished name of the mailing list.

- ```
  objectClass: groupOfUniqueNames
  objectClass: inetMailGroup
  objectClass: inetLocalMailRecipient
  objectClass: inetMailGroupManagement
  objectClass: nsManagedMailList
  ```

`groupOfUniqueNames` is the core object class for all mailing list entries. Overlaying with the mail service `objectclassesinetMailGroup`, `inetLocalMailRecipient`, and the Delegated Administrator service object classes `inetMailGroupManagement` makes the entry a mailing list for use by the Messaging Server.

`inetMailGroup` specifies attributes for mailing lists.

`inetLocalMailRecipient` provides internal routing attributes.
  ```

Note
The `inetLocalMailRecipient` object class is intended to support SMTP message transfer agents in routing RFC 822-based email within a private enterprise only and is not to be used in the process of routing email across the public Internet.

`inetMailGroupManagement` specifies attributes for managing a mailing list.

`nsManagedMailList` provides Delegated Administrator support attributes for mailing lists.

groupOfUniqueNames

- `cn: Dreamteam`
  `cn(commonname)` is the mailing list's name.

- `uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp`
  `uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp`
  `uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp`
  `uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp`
  `uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp`

These are the members of the mailing list that can be resolved to a user in this directory.

inetMailGroup

- `inetMailGroupStatus: active`
  Current status of the mailing list: `active`, `inactive`, or `deleted`. A missing value implies a status is `active`. An illegal value is treated as `inactive`.

- `dataSource: users 12/3/12`
  Text field to store a tag or identifier.

- `mgrpRFC822MailMember: west@florizel.com`
  `mgrpRFC822MailMember: robertson@florizel.com`
  External members of the mailing list.

inetLocalMailRecipient

- `mail: dreamteam@sesta.com`
  The mailing list's advertised email address (RFC 822 format).
- `mailAlternateAddress: thegreatest@sesta.com`
  Alternate RFC822 email address of this mailing list.

- `mailHost: manatee.siroe.com`
  Fully-qualified hostname of the MTA server that is the final SMTP destination of messages to this mailing list.

nsManagedMailList

- `nsNumUsers: 7`
  Current number of user entries.

- `nsMaxUsers: 1000`
  Maximum user entries.

Attribute Values Format

Several attributes, such as `moderator` and `mailDeliveryURL`, require that you specify user addresses or file names as URLs. When preceded by `ldap:///`, the entry is taken as an LDAP entry with the remaining value treated as the distinguished name of the entry. For example:

```
moderator: ldap:///uid=cox,ou=people,o=sesta,o=isp
```

If attribute-value pairs span two lines, the second line must start with a blank space. When preceded by `mailto: ofanning@sesta.com`, the entry is interpreted as a mail address. When preceded by `file:///`, the entry is interpreted as a file. For example:

```
mailDeliveryURL: file:///home/dogboy/dr_j/mail_archive.htm
```

Assigning Mailing List Owners

Mailing list owners can add or delete members to the list. To change an owner of a mailing list, assign a DN to the owner attribute. There can be more than one owner for the mailing list, but owners must have DNs in the same directory as the mailing list. Following is an example modification statement and LDIF record.

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: owner
owner:  uid=baylor,ou=People,o=sesta.com,o=isp
```

The following is an example LDIF Record for a Mailing List with an Owner:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: users 12/3/12
inetMailGroupStatus: active
nsNumUsers:
nsMaxUsers: 1000
owner:  uid=baylor,ou=People,o=sesta.com,o=isp
```

Adding Mailing List Members

Add internal members (members with resolvable DNs) by assigning their DN to the attribute `uniqueMember`. Add external members by assigning their email address to the attribute `mgrpRFC822MailMember`.

The following example LDIF code shows how to add an internal and external user:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: uniqueMember
uniqueMember: uid=russell,ou=People,o=sesta.com,o=isp
-
add: mgrpRFC822MailMember
mgrpRFC822MailMember: chamberlain@varrius.com
```

The following is an example LDIF Record for a Mailing List with Added Members:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
uniqueMember: uid=russell,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: chamberlain@varrius.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: users 12/3/12
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
owner: uid=baylor,ou=People,o=sesta.com,o=isp
```

Creating Posting Restrictions on Mailing Lists

Incoming mail to a mailing list can be restricted by domain or user. The restriction attributes are as follows:

- `mgrpAllowedBroadcaster` specifies addresses authorized to send messages to the mailing list. If you do not include `mgrpAllowedBroadcaster` in the LDAP entry, the list is unrestricted and anyone can submit a message to it. The envelope `From:` address must match one of the addresses in the permitted list before the MTA routes the message to a list of members.

- `mgrpDisallowedBroadcaster` specifies addresses restricted from posting messages to the list. The sender's address is compared against those in this attribute. If there is a match then the message is rejected.
- `mgrpAllowedDomain` specifies the domain names from which users are authorized to post messages to the mailing list. The wildcard character is "*". Using the wildcard character, you can optionally replace a sub-domain to authorize the entire DNS hierarchy below a given top or sub-domain.
- `mgrpDisallowedDomain` defines the domain names from which users cannot post messages to the mailing list.

> **Note**
> DN values for `mgrpAllowedBroadcaster` and `mgrpDisallowedBroadcaster` must have the prefix `ldap:///` or `mailto:`. Refer to Attribute Values Format.

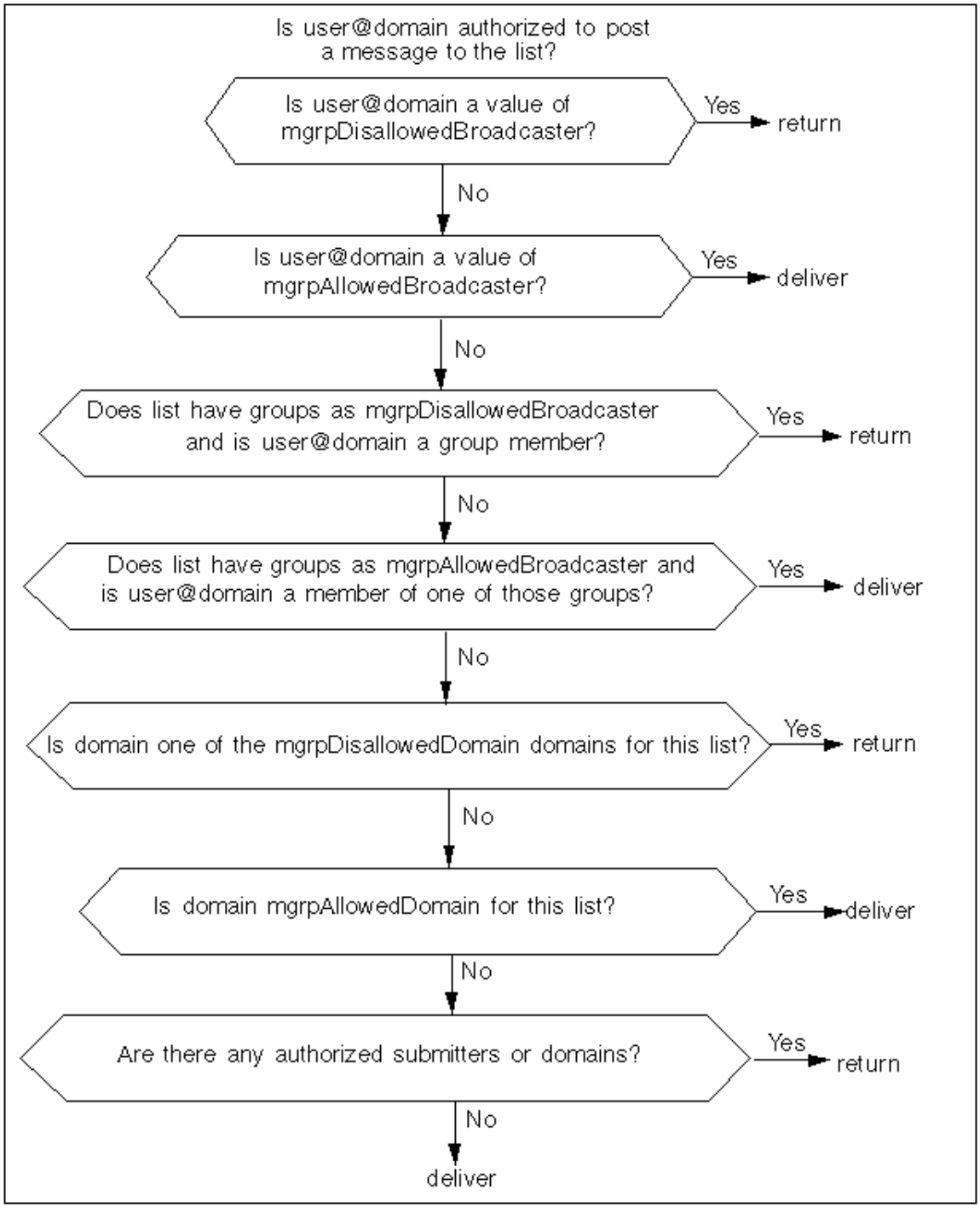
Precedence Rules

The MTA uses the following precedence rules when deciding whether to accept the message for further processing or not (all rules use the envelope "From:" address when looking for a match):

1. If `mgrpDisallowedBroadcaster` is set, there must not be a match between this value and the sender's mail attribute or `mailAlternateAddress` attribute of any DN listed in the form of an `ldap:///`*DN* address, or there must not be a match with the RFC-822 address listed in the form of a `mailto:`*RFC-822* address.
2. If `mgrpAllowedBroadcaster` exists in the LDAP entry, the sender's address must match either the mail attribute or `mailAlternateAddress` attribute of any DN listed in the form of an `ldap:///`*DN* address or must match the RFC-822 address listed in the form of a `mailto:` *RFC-822* address.
3. If `mgrpDisallowedDomain` exists in the LDAP entry, then sender's domain must not match the domain(s) listed in the `mgrpDisallowedDomain` attribute.
4. If `mgrpAllowedDomain` exists in the LDAP entry, then the sender's domain must match the domain(s) listed in the `mgrpAllowedDomain` attribute.

The following diagram shows the access control process.

Access Control Process



The following example LDIF allows users on the `sesta.com` domain to send messages to the mailing list, but blocks users on all other domains from sending messages. It also blocks internal mail from the email address `barry@sesta.com` but allows mail from email address `barkley@florizel.com`.

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgrpAllowedDomain
mgrpAllowedDomain: sesta.com
-
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgrpAllowedBroadcaster
mgrpAllowedBroadcaster: mailto: barkley@florizel.com
-
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgrpDisallowedBroadcaster
mgrpDisallowedBroadcaster: ldap:///cn=barry,ou=people,o=sesta.com,o=isp
```

The following is an example Mailing List LDIF Record with Delivery Restrictions:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: users 12/3/12
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mgrpAllowedDomain: sesta.com
mgrpAllowedBroadcaster: mailto: barkley@florizel.com
mgrpDisallowedBroadcaster: ldap:///cn=barry,ou=people,o=sesta.com,o=isp
```

Assigning Mailing List Moderators

A mailing list moderator is a user who receives a mailing list message before all other members, then forwards it to the rest of the members if desired. Any message submitted to the mailing list will go to the moderator instead of the mailing list members. Set a valid DN or email address to the attribute moderator. Multiple moderators are allowed.

Moderators are created by setting `mgrpModerator` to an RFC 822 email address or a DN in URL format.

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgrpModerator
mgrpModerator: ldap:///uid=baylor,ou=People,o=sesta.com,o=isp
```

The following is an example Mailing List LDIF Record with Moderator:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mgrpModerator: ldap:///uid=baylor,ou=People,o=sesta.com,o=isp
```

Enabling, Disabling, and Deleting Mailing Lists

Mailing lists can be enabled, temporarily disabled, or deleted by setting `inetMailGroupStatus` to active, inactive, or deleted. A disabled mailing list can be activated by resetting `inetMailGroupStatus` to active. Missing value implies status is active. An illegal value is treated as inactive.

The following LDIF code disables the mailing list:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: inetMailGroupStatus
inetMailGroupStatus: inactive
```

The following is an example LDIF Record with Mailing List Disabled:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
nsNumUsers: 7
nsMaxUsers: 1000
inetMailGroupStatus: inactive
```

Archiving Messages to a File

Archive mailing list messages by setting `mailDeliveryFileURL` to a URL file. The example below saves messages to `dreamteam@sesta.com` to `/home/dreamteam/mail_archive.log`

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mailDeliveryFileURL
mailDeliveryFileURL: [file:///home/dreamteam/mail_archive.log]
```

The following is an example Mailing List LDIF Record with Archive Attribute:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mailDeliveryFileURL: [file:///home/dreamteam/mail_archive.log]
```

Specifying the Mailing List Request Addresses

Specify the mailing list subscription request address with the `mgrpRequestsTo` attribute. An example of a subscription request is `dreamteam-request@sesta.com` Only internal addresses are allowed, and these must be in URL format.

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgrpRequestsTo
mgrpRequestsTo: uid=baylor,ou=People,o=sesta.com,o=isp
```

The followingt is an example Mailing List LDIF Record with Subscription Request Attribute:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mgrpRequestsTo: uid=baylor,ou=People,o=sesta.com,o=isp
```

Enabling Mailing List Member Visibility

Mailing list members are typically visible through the iPlanet Console, iPlanet Delegated Administrator for Messaging, or the SMTP `EXPN` command. You specify visibility of mailing list members setting the attribute `mgmanMemberVisibility`. The possible values for this attribute are ANYONE (anyone in the world can view), ALL (anyone in the directory can view), NONE (only owner can view).

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgmanMemberVisibility
mgmanMemberVisibility: ALL
```

The following is an example Mailing List LDIF Record with Archive Attribute:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mgmanMemberVisibility: ALL
```

Making Mailing Lists Joinable

You can specify who may join a mailing list by setting the attribute `mgmanJoinability`. The possible values for this attribute are ANYONE (anyone in the world can join), ALL (anyone in the directory can join), NONE (no additional members can join).

```
dn: cn=dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mgmanJoinability
mgmanJoinability:  ALL
```

The following is an example LDIF Record for a Joinable Mailing List:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mgmanJoinability:  All
```

Creating Dynamic Mailing Lists

Messaging Server supports both static and dynamic mailing lists. Unlike static mailing lists, where members of the list are specified by using `uniquemember` and `mgrprfc822mailmember` attributes, dynamic mailing list members are specified using an LDAP search filter (RFC-2254). The LDAP filter is set in the `mgrpDeliverTo` attribute in the `inetMailGroup` objectclass.

The example below shows a mailing list consisting of static members and members determined with an LDAP search filter. The filter below includes as members in `o=sesta.com,o=isp` who also have the attribute value-pair `city=tokyo`. You should make sure that the attributes used in the LDAP search filter are indexed; otherwise, the evaluating membership of dynamic lists will be both time consuming and stress the directory server.

The following is an example LDIF Record for a Dynamic Mailing List:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mgrpdeliverto: ldap:///o=sesta.com,o=isp??sub?
(&(objectclass=inetMailUser)(city=tokyo))
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mgmanJoinability:  All
```

Note
iPlanet Messaging Server also supports dynamic lists based on the attribute memberURL from the objectclass groupofurls. Netscape Directory Server 4.x allows creating of dynamic groups using this attribute and messaging server can take advantage of any groups that may have already been defined using memberURL.

Limiting Received Message Size

To limit the size of a message that a user or group can receive, use the attribute `mailMsgMaxBlocks`. Message size is set in MTA block size (default is 1024 bytes). Note that this attribute overrides `mailDomainMsgMaxBlocks`.

The example below sets a limit of one megabyte.

```
dn: cn=dreamteam,ou=groups,o=sesta.com,o=isp
changetype: modify
add: mailMsgMaxBlocks
mailMsgMaxBlocks:  1000
```

The following is an example LDIF Record for a Joinable Mailing List:

```
dn: cn=Dreamteam,ou=groups,o=sesta.com,o=isp
cn: Dreamteam
objectClass: groupOfUniqueNames
objectClass: inetMailGroup
objectClass: inetLocalMailRecipient
objectClass: inetMailGroupManagement
objectClass: nsManagedMailList
uniqueMember: uid=baylor,ou=People,o=sesta.com,o=isp
uniqueMember: uid=bird,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jordan,ou=People,o=sesta.com,o=isp
uniqueMember: uid=jabbar,ou=People,o=sesta.com,o=isp
uniqueMember: uid=magic,ou=People,o=sesta.com,o=isp
mgrpRFC822MailMember: west@florizel.com
mgrpRFC822MailMember: robertson@florizel.com
mail: dreamteam@sesta.com
mailAlternateAddress: thegreatest@sesta.com
mailHost: manatee.siroe.com
dataSource: @(#)ims50users.sh 1.5a 02/3/00
inetMailGroupStatus: active
nsNumUsers: 7
nsMaxUsers: 1000
mailMsgMaxBlocks: 1000
```

Chapter 14. Using Predefined Channels

Using Predefined Channels

When you first install Messaging Server, several channels are already defined (see Predefined Channels). This information describes how to use predefined channel definitions in the MTA.

If you have not already read About MTA Services and Configuration, you should do so before reading this information. See Configuring Rewrite Rules for information about configuring the rewrite rules in the `imta.cnf` file.

Topics:

- Predefined Channels
- To Deliver Messages to Programs Using the Pipe Channel
- To Configure the Native (/var/mail) Channel
- To Temporarily Hold Messages Using the Hold Channel
- The Conversion Channel
- Character Set Conversion and Message Reformatting

The `defaults` channel is described in http://msg.wikidoc.info/index.php/Channel_configuration.

Predefined Channels

The following table lists some of the predefined channels.

Some Predefined Channels

Channel	Definition
defaults	Used to specify which keywords are defaults for various channels. See http://msg.wikidoc.info/index.php/Channel_configuration.
l	UNIX only. Used to make routing decisions and for submitting mail using UNIX mail tools.
ims-ms	Performs final delivery of mail to the local store.
native	UNIX only. Delivers mail to /var/mail. (Messaging Server does not support /var/mail access. User must use UNIX tools to access mail from the /var/mail store.)
pipe	Used to perform delivery via a site-supplied program or script. Commands executed by the pipe channel are controlled by the administrator by using the imsimta program interface.
reprocessprocess	These channels are used for deferred, offline message processing. The reprocess channel is normally invisible as a source or destination channel. The process channel is visible like other MTA channels.
defragment	Provides the means to reassemble MIME fragmented messages.
conversion	Performs body-part-by-body-part conversions on messages flowing through the MTA.
bitbucket	Used for messages that need to be discarded.
inactive/deleted	Used to process messages for users who have been marked as inactive or deleted in the directory. Typically, bounces the message and returns custom bounce message to the sender of the message.
hold	Used to hold messages for users, for example, when a user is migrated from one mail server to another.
sms	Provides support for one-way email to an SMS gateway.
tcp_local tcp_intranet tcp_auth tcp_submit tcp_tas	Implements SMTP over TCP/IP. The multithreaded TCP SMTP channel includes a multithreaded SMTP server that runs under the control of the Dispatcher. Outgoing SMTP mail is processed by the channel program tcp_smtp_client, and runs as needed under the control of the Job Controller. tcp_local receives inbound messages from remote SMTP hosts. Depending on whether you use a smarthost/firewall configuration, either sends outbound messages directly to remote SMTP hosts or sends outbound messages to the smarthost/firewall system. Sometimes tcp_local gets mail from remote SMTP hosts via proxy or firewall. tcp_local is also sometimes used for internal relay activities. tcp_intranet receives and sends messages within the intranet. tcp_auth is used as a switch channel for tcp_local; authenticated users switch to the tcp_auth channel to avoid relay-blocking restrictions. tcp_submit accepts message submissions, usually from user agents, on the reserved submission port 587 (see RFC 2476). tcp_tas is a special channel used by sites doing Unified Messaging.

To Deliver Messages to Programs Using the Pipe Channel

Users might want incoming mail passed to a program instead of to their mailbox. For example, users might want their incoming mail sent to a mail sorting program. The pipe channel performs delivery of messages using per-user, site-supplied programs.

To facilitate program delivery, you must first register programs as able to be invoked by the pipe

channel. Do this by using the `imsimta` utility. This utility gives a unique name to each command that you register as able to be invoked by the `pipe` channel. End users can then specify the method name as a value of their `mailprogramdeliveryinfo` LDAP attribute.

For example, to add a UNIX command `myprocmail` as a program that can be invoked by a user, you would first register the command by using the `imsimta program` utility as shown in the following example. This example registers a program called `myprocmail` that executes the program `procmail` with the arguments `-d` *username* and executes as the user:

```
imsimta program -a -m myprocmail -p procmail -g "-d %s" -e user
```

Make sure the executable exists in the `programs` directory *msg-svr-base*`/data/site-programs`. Make sure also that the execute permissions are set to "others."

To enable a user to access the program, the user's LDAP entry must contain the following attributes and values:

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

Alternative delivery programs must conform to the following exit code and command-line argument restrictions:

Exit Code Restrictions. Delivery programs invoked by the `pipe` channel must return meaningful error codes so that the channel knows whether to dequeue the message, deliver for later processing, or return the message.

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 the MTA 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`.

Command Line Arguments. Delivery programs can have any number of fixed arguments as well as the variable argument, `%s`, representing the user name for programs executed by the user or username+domain for programs executed by the postmaster, "inetmail." For example, the following command line delivers a recipient's mail using the program `procmail`:

```
/usr/lib/procmail -d %s
```

To Configure the Native (/var/mail) Channel

An option file can be used to control various characteristics of the native channel. This native channel option file must be stored in the MTA configuration directory and named `native_option` (for example, *msg-svr-base*`/config/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* can be either a string or an integer, depending on the option's requirements.

Local Channel Options

Options	Descriptions
FORCE_CONTENT_LENGTH (0 or 1; UNIX only)	If FORCE_CONTENT_LENGTH=1, then the MTA 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.
FORWARD_FORMAT (string)	Specifies the location of the users' .forward files. The string %u indicates that it is substituted in each user id. The string %h indicates that it is substituted in each user's home directory. The default behavior, if this option is not explicitly specified, corresponds to: FORWARD_FORMAT=%h/.forward
REPEAT_COUNT (integer) SLEEP_TIME (integer)	In case the user's new mail file is locked by another process when the MTA tries to deliver the new mail, these options provide a way to control the number and frequency of retries the native channel program should attempt. If the file cannot be opened after the number of retries specified, the messages remain in the native queue and the next run of the native channel attempts to deliver the new messages again. The REPEAT_COUNT option controls how many times the channel programs 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).
SHELL_TIMEOUT (integer)	Controls the length of time in seconds the channel waits for a user's shell command in a .forward to complete. Upon such time-outs, the message are returned to the original sender with an error message resembling "Time-out waiting for *user*'s shell command *command* to complete." The default is 600 (10 minutes).
SHELL_TMPDIR (directory-specific)	Controls the location where the local channel creates its temporary files when delivering to a shell command. By default, such temporary files are created in users' home directories. Using this option, the administrator may instead choose the temporary files to be created in another (single) directory. For example: SHELL_TMPDIR=/tmp

To Temporarily Hold Messages Using the Hold Channel

The hold channel is used to hold the messages of a recipient temporarily prevented from receiving new messages. Messages may be held because users' names are being changed or their mailboxes are being moved from one mailhost or domain to another. There may also be other reasons to temporarily hold messages.

When messages are to be held, they are directed to the hold channel, in the *msg-svr-base*/queue/hold

directory, using the same mechanism used to direct messages to the reprocess channel. In this way, the envelope To: addresses are unchanged. The messages are written to the hold channel queue, in the *msg-svr-base*`/queue/hold directory`, as *ZZxxx*`.HELD` files. This prevents them from being seen by the job controller, and thus they are "held." Use the `imsimta qm dir -held` command to view a list of `.HELD` files. These messages can be selected and released by using the `imsimta qm release` command. Releasing them changes their name to *ZZxxx*`.00` and informs the job controller. The messages are then processed by the master program associated with the hold channel, `reprocess.exe`. Thus the message (and the To: addresses) are processed by using the normal rewriting machinery.

See the `imsimta qm` documentation in *Messaging Server Administration Reference* for more information.

The Conversion Channel

The `conversion` channel enables you to perform arbitrary body part-by-body part processing on specified messages flowing through the MTA. (Note that a body part is different than a message in that a message can contain multiple body parts as, for instance, in an attachment. Also, body parts are specified and delineated by MIME headers.) This processing can be done by any site-supplied programs or command procedures and can do such things such as convert text or images from one format to another, virus scanning, language translation, and so forth. Various message types of the MTA traffic are selected for conversion, and specific processes and programs can be specified for each type of message body part.

The prerequisite for using the `conversion` channel is understanding the concept of channels (see Channels). For supplemental information on virus scanning using the `conversion` channel, refer to Virus Screening with the iPlanet Messaging Server Conversion Channel. You can follow the sample scan_sh script to implement scanning with the `conversion` channel.

Implementing the conversion channel consists of the following high-level steps:

1. Selecting message traffic for processing
2. Specifying how different messages will be processed. These procedures are described later.

> **Note**
> A default conversion channel is automatically created in the MTA configuration file (`imta.cnf`). This channel can be used as is and requires no modification.

This section consists of the following sections:

- MIME Overview
- Selecting Traffic for Conversion Processing
- To Control Conversion Processing
- To Bounce, Delete, Hold, Retry Messages Using the Conversion Channel Output
- Conversion Channel Example
- Automatic Arabic Character Set Detection
- To Automatically Detect Arabic Character Sets

MIME Overview

The conversion channel makes extensive use of the MIME (Multipurpose Internet Mail Extensions) header lines. Knowledge of message construction and MIME header fields is required. For complete information on MIME, refer to http://www.faqs.org/rfcs/. A short overview of MIME is presented here for convenience.

Message Construction

A simple message consists of a header and a body. The header is at the top of the message and contains certain control information such as date, subject, sender, and recipient. The body is everything after the first blank line after the header. MIME specifies a way to construct more complex messages which can contain multiple body parts, and even body parts nested within body parts. Messages like these are called multi-part messages, and, as mentioned earlier, the conversion channel performs body part-by-body part processing of messages.

MIME Headers

The MIME specification defines a set of header lines for body parts. These include `MIME-Version`, `Content-type`, `Content-Transfer-Encoding`, `Content-ID`, and `Content-disposition`. The conversion channel uses the `Content-type` and `Content-disposition` headers most frequently. The following shows an example of some MIME header lines:

```
Content-type: APPLICATION/wordperfect5.1;name=Poem.wpc
Content-transfer-encoding: BASE64
Content-disposition: attachment; filename=Poem.wpc
Content-description: "Project documentation Draft1 wordperfect format"
```

Note
MIME header lines are not the same as general, non-MIME header lines such as `To:`, `Subject:` and `From:`. Basically, for Conversion channel discussion, MIME header lines start with the string `Content-`.

Content-type Header

The MIME `Content-Type` header describes the content of the body-part. The following shows an example of a `Content-Type` header format:

`Content-type:` *type*/*subtype*; *parameter*=*value*; *parameter*=*value...*

type describes the type of content of the body part. Examples of type are `Text`, `Multipart`, `Message`, `Application`, `Image`, `Audio`, and `Video`.

subtype further describes content type. Each `Content-type` has its own set of subtypes. For examples: `text/plain`, `application/octet-stream`, and `image/jpeg`. Content Subtypes for MIME mail are assigned and listed by the IANA (Internet Assigned Numbers Authority). A copy of the list is at http://www.iana.org/assignments/media-types.

parameter is specific to Content-type/subtype pairs. For example, the `charset` and the `name` parameters are shown below:

```
Content-type: text/plain; charset=us-ascii
Content-type: application/msword; name=temp.doc
```

The `charset` parameter specifies a character set for a textual message. The `name` parameter gives a suggested file name to be used if the data were to be written to a file.

Note
`Content-Type` values, `subtypes`, and parameter names are case-insensitive.

Content-disposition Header

The MIME `Content-disposition` header provides presentation information for the body-part. It is often added to attachments specifying whether the attachment body part should be displayed (`inline`) or presented as a file name to be copied (`attachment`). The `Content-disposition` header has the following format:

`Content-disposition:` *disposition_type; parameter=value;parameter=value...*

disposition_type is usually `inline` (display the body part) or `attachment` (present as file to save.) `Attachment` usually has the parameter `filename` with a value specifying the suggested name for the saved file.

For details on the `Content-disposition` header, refer to RFC2183.

Selecting Traffic for Conversion Processing

Unlike other MTA channels, the conversion channel is not normally specified in an address or MTA rewrite rule. Instead, messages are sent through the conversion channel if they meet the criteria specified in the `CONVERSIONS` mapping table (the name of the `mappings` file is specified by the parameter `IMTA_MAPPING_FILE` in the `imta_tailor` file. The default is *msg-srv-base*`/conversions`). Entries to the table have the following format:

```
IN-CHAN=source-channel;OUT-CHAN=destination-channel;CONVERT Yes/No
```

As the MTA processes each message it probes the `CONVERSIONS` mapping table (if one is present). If the *source-channel* is the channel from which the message is coming and *destination-channel* is the channel to which the message is going, then the action following `CONVERT` is taken. `Yes` means the MTA diverts the message through the conversion channel on its way to its *destination-channel*. If no match is found, or if `No` was specified, the message is queued to the regular destination channel.

If you route messages to the conversion channel by using conversion mappings, your other mappings should continue to work. However, if you use rewrite rules to route messages to the conversion channel, you might have to adjust your mappings to accommodate what you've done.

> **Note**
> An address of the form `user@conversion.`*localhostname* or `user@conversion` will be routed through the conversion channel, regardless of the `CONVERSIONS` mapping table.

The following example routes all non-internal messages--messages originating from, or destined to, the Internet--through the conversion channel.

```
CONVERSIONS

   IN-CHAN=tcp_local;OUT-CHAN=*;CONVERT   Yes
   IN-CHAN=*;OUT-CHAN=tcp_local;CONVERT   Yes
```

The first line specifies that messages coming from the `tcp_local` channel will be processed. The second line specifies that messages going to the `tcp_local` channel will also be processed. The `tcp_local` channel handles all messages going to and coming from the Internet. Since the default is to not go through the conversion channel, any other messages won't go through the conversion channel.

Note that this is a very basic table, and that it might not be sufficient for a site with a more customized

configuration, for example, one using multiple outbound-to-the-Internet `tcp_*` channels, or using multiple inbound-from-the-Internet `tcp_*` channels.

To Control Conversion Processing

This section describes how to control conversion processing.

When a message is sent to the conversion channel, it is processed body part-by-body part. Processing is controlled by the MTA `conversions` file, which is specified by the `IMTA_CONVERSION_FILE` option in the `imta_tailor` file (default: *msg-svr-base*`/conversions`). The `conversions` file consists of line-separated entries that 1) qualify which types of body parts will be processed, and 2) how they will be processed.

Each entry consists of one or more lines containing one or more *name*=*value* parameter clauses. Where the name in the > parameter clauses is one of the parameters in LDAP URL Substitution Sequences. The values in the parameter clauses conform to MIME conventions. 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 back slash (\) continuation character. Entries are terminated either by a line that does not end in a semicolon, one or more blank lines, or both.

The following is a simple example of a `conversion` file entry:

conversions File Entry

```
out-chan=ims-ms; in-type=application; in-subtype=wordperfect5.1;
  out-type=application; out-subtype=msword; out-mode=block;
  command="/usr/bin/convert -in=wordp -out=msword 'INPUT_FILE'
'OUTPUT_FILE&rsquo;"
```

The clauses `out-chan=ims-ms; in-type=application; in-subtype=wordperfect5.1` qualify the body part. That is, they specify the type of part to be converted. 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 `command=` or `delete=` clause is performed, and the* `out-*` parameters are set.

If no match occurs, then the part is matched against the next `conversions` file entry. Once all body parts have been scanned and processed (assuming there is a qualifying match), then the message is sent onwards to the next channel. If there are no matches, no processing occurs, and the message is sent to the next channel.

`out-chan=ims-ms` specifies that only message parts destined for the `ims-ms` channel will be converted. `in-type=application` and `in-subtype=wordperfect5.1` specifies that the MIME `Content-type` header for the message part must be `application/wordperfect5.1`.

Message parts can be further qualified with additional `in-*` parameters. (See Conversion Parameters.) The entry above will trigger conversion actions on a message part which has the following MIME header lines:

```
Content-type: APPLICATION/wordperfect5.1;name=Draft1.wpc
Content-transfer-encoding: BASE64
Content-disposition: attachment; filename=Draft1.wpc
Content-description: "Project documentation Draft1 wordperfect format"
```

After the three `conversion` file qualifying parameters in conversions File Entry, the next two parameters, `out-type=application` and `out-subtype=msword`, specify replacement MIME header lines to be attached to the "processed" body part. `out-type=application` and `out-subtype=msword` specify that the MIME `Content-type/subtype` of the outgoing message be `application/msword`.

Note that since the `in-type` and `out-type` parameters are the same, `out-type=application` is not necessary since the conversion channel defaults to the original MIME labels for outgoing body parts. Additional MIME labels for outgoing body parts can be specified with additional output parameters.

`out-mode=block` (conversions File Entry) specifies the file type that the site-supplied program will return. In other words, it specifies how the file will be stored and how the conversion channel should be read back in the returned file. For example, an html file is stored in text mode, while an `.exe` program or a zip file is stored in block/binary mode. Mode is a way of describing that the file being read is in a certain storage format.

The final parameter in conversions File Entry specifies the action to take on the body part:

```
command="/usr/bin/convert -in=wordp -out=msword 'INPUT_FILE' 'OUTPUT_FILE'"
```

The `command=` parameter specifies that a program will execute on the body part. `/usr/bin/convert` is the hypothetical command name; `-in=wordp` and `-out=msword` are hypothetical command line arguments specifying the format of the input text and output text; `INPUT_FILE` and `OUTPUT_FILE` are conversion channel environmental variables (see To Use Conversion Channel Environmental Variables program should store its converted body part.

> **Note**
> Envelope originator and recipient information is now provided as x-envelope-from and x-envelope-to fields respectively when a file containing the outer message header is requested by a regular conversion entry.

Instead of executing a command on the body part, the message part can simply be deleted by substituting `DELETE=1` in place of the `command` parameter.

> **Note**
> Whenever the `conversions` file is modified, you must recompile the configuration (see Compiling the MTA Configuration).

Conversion Channel Information Flow

The flow of information is as follows: a message containing body parts comes into the conversion channel. The conversion channel parses the message, and processes the parts one by one. The conversion channel then qualifies the body part, that is, it determines if it should be processed or not by comparing its MIME header lines to the *qualifying parameters* (Conversion Parameters). If the body part qualifies, the conversion processing commences.

If MIME or body part information is to be passed to the conversion script, it is stored in an environmental variable (To Use Conversion Channel Environmental Variables) as specified by *information passing parameters* (Conversion Parameters).

At this point, an action specified by an *action parameter,* (Conversion Parameters) is taken on the body part. Typically the action is that the body part be deleted or that it be passed to a program wrapped in a

script. The script processes the body part and then sends it back to the conversion channel for reassembling into the post-processed message. The script can also send information to the conversion channel by using the conversion channel *output options* (Conversion Channel Output Options). This can be information such as new MIME header lines to add to the output body part, error text to be returned to the message sender, or special directives instructing the MTA to initiate some action such as bounce, delete, or hold a message.

Finally, the conversion channel replaces the header lines for the output body part as specified by the *output parameters* (Conversion Parameters).

To Use Conversion Channel Environmental Variables

When operating on message body parts, it is often useful to pass MIME header line information, or entire body parts, to and from the site-supplied program. For example, a program may require `Content-type` and `Content-disposition` header line information as well as a message body part. Typically a site-supplied program's main input is a message body part which is read from a file. After processing the body part, the program will need to write it to a file from which the conversion channel can read it. This type of information passing is done by using conversion channel environmental variables.

Environmental variables can be created in the `conversions` file using the `parameter-symbol-*` parameter or by using a set of pre-defined conversion channel environmental variables (see To Use Conversion Channel Output Options).

The following `conversions` file entry and incoming header show how to pass MIME information to the site-supplied program using environment variables.

`conversions` file entry:

```
in-channel=*; in-type=application; in-subtype=*;
  parameter-symbol-0=NAME; parameter-copy-0=*;
  dparameter-symbol-0=FILENAME; dparameter-copy-0=*;
  message-header-file=2; original-header-file=1;
  override-header-file=1; override-option-file=1;
  command="/bin/viro-scan500.sh "INPUT_FILE' "OUTPUT_FILE'"
```

Incoming header:

```
Content-type: APPLICATION/msword; name=Draft1.doc
Content-transfer-encoding: BASE64
Content-disposition: attachment; filename=Draft1.doc
Content-description: "Project documentation Draft1 msword format"
```

`in-channel=*; in-type=application; in-subtype=*` specify that a message body part from any input channel of type `application` will be processed.

`parameter-symbol-0=NAME` specifies that value of the `Content-type` parameter `name`, if present (`Draft1.doc` in our example), be stored in an environment variable called `NAME`.

`parameter-copy-0=*` specifies that all `Content-type` parameters of the input body part be copied to the output body part.

`dparameter-symbol-0=FILENAME` specifies that the value of the `Content-disposition` parameter `filename` (`Draft1.doc` in our example), be stored in an environment variable called

FILENAME.

dparameter-copy-0=* specifies that all Content-disposition parameters of the input body part be copied to the output body part.

message-header-file=2 specifies that the original header of the message as a whole (the outermost message header) be written to the file specified by the environment variable MESSAGE_HEADERS.

original-header-file=1 specifies that the original header of the enclosing MESSAGE/RFC822 part are written to the file specified by the environment variable INPUT_HEADERS.

override-header-file=1 specifies that MIME headers are read from the file specified by environmental variable OUTPUT_HEADERS, overriding the original MIME header lines in the enclosing MIME part. $OUTPUT_HEADERS is an on-the-fly temporary file created at the time conversion runs. A site-supplied program would use this file to store MIME header lines changed during the conversion process. The conversion channel would then read the MIME header lines from this file when it re-assembles the body part. Note that only MIME header lines can be modified. Other general, non-MIME header lines cannot be cannot be altered by the conversion channel.

override-option-file=1 specifies that the conversion channel read *conversion channel options* from the file named by the OUTPUT_OPTIONS environmental variable. See To Use Conversion Channel Output Options.

command="msg-svr-base/bin/viro-scan500.sh" specifies the command to execute on the message body part.

Conversion Channel Environment Variables

Environment Variable	Description
ATTACHMENT_NUMBER	Attachment number for the current part. This has the same format as the ATTACHMENT-NUMBER conversion match parameter.
CONVERSION_TAG	The current list of active conversion tags. This corresponds to the TAG conversion match parameter.
INPUT_CHANNEL	The channel that enqueued the message to the conversion channel. This corresponds to the IN-CHANNEL conversion match parameter.
INPUT_ENCODING	Encoding originally present on the body part.
INPUT_FILE	Name of the file containing the original body part. The site-supplied program should read this file.
INPUT_HEADERS	Name of the file containing the original header lines for the body part. The site-supplied program should read this file.
INPUT_TYPE	MIME `Content-type` of the input message part.
INPUT_SUBTYPE	MIME content subtype of the input message part.
INPUT_DESCRIPTION	MIME `content-description` of the input message part.
INPUT_DISPOSITION	MIME `content-disposition` of the input message part.
MESSAGE_HEADERS	Name of the file containing the original outermost header for an enclosing message (not just the body part) or the header for the part's most immediately enclosing MESSAGE/RFC822 part. The site-supplied program should read this file.
OUTPUT_CHANNEL	The channel the message is headed for. This corresponds to the OUT-CHANNEL conversion match parameter.
OUTPUT_FILE	Name of the file where the site-supplied program should store its output. The site-supplied program should create and write this file.
OUTPUT_HEADERS	Name of the file where the site-supplied program should store MIME header lines for an enclosing part. The site-supplied program should create and write this file. Note that file should contain actual MIME header lines (not `option=value` lines) followed by a blank line as its final line. Note also that only MIME header lines can be modified. Other general, non-MIME header lines cannot be cannot be altered by the conversion channel.
OUTPUT_OPTIONS	Name of the file from which the site-supplied program should read conversion channel options. See To Use Conversion Channel Output Options.
PART_NUMBER	The part number for the current part. This has the same format as the PART-NUMBER conversion match parameter.
PART_SIZE	The size in bytes of the part being processed.

Mail Conversion Tags

Mail conversion tags are special tags which are associated with a particular recipient or sender. When a message is being delivered, the tag is visible to the conversion channel program, which may make use of it for special processing. Conversion tags are stored in the LDAP directory.

Mail conversion tags could be used as follows: the administrator can set up selected users with a mail conversion tag value of `harmonica`. The administrator then has a conversion channel setup which, when processing that mail, will detect the presence of the tag and the value of `harmonica`. When that

happens, the program will perform some arbitrary function.

Mail conversion tags can be set on a per user or a per domain basis. The recipient LDAP attribute at the domain level is `MailDomainConversionTag` (modifiable with the MTA option {LDAP_DOMAIN_ATTR_CONVERSION_TAG}}). At the user level it is `MailConversionTag` (modifiable with the MTA option `LDAP_CONVERSION_TAG`). Both of these attributes can be multivalued with each value specifying a different tag. The set of tags associated with a given recipient is cumulative, that is, tags set at the domain level are combined with tags set at the user level.

Sender-based conversion tags can be set with the MTA options `LDAP_SOURCE_CONVERSION_TAG` and `LDAP_DOMAIN_ATTR_SOURCE_CONVERSION_TAG,` which specify user and domain level LDAP attributes respectively for conversion tags associated with these source address. There is no default attribute for either of these options.

Two new actions are available to system Sieves: `addconversiontag` and `setconversiontag`. Both accept a single argument: A string or list of conversion tags. `addconversiontag` adds the conversion tag(s) to the current list of tags while `setconversiontag` empties the existing list before adding the new ones. Note that these actions are performed very late in the game so `setconversiontag` can be used to undo all other conversion tag setting mechanisms. These allow you put conversion tags in the Sieves filters.

The Sieve envelope test accepts `conversiontag` as an envelope field specifier value. The test checks the current list of tags, one at a time. Note that the `:count` modifier, if specified, allows checking of the number of active conversion tags. This type of envelope test is restricted to system Sieves. Also note that this test only sees the set of tags that were present prior to Sieve processing--the effects of `setconversiontag` and `addconversiontag` actions are not visible.

Including Conversion Tag Information in Various Mapping Probes

A new MTA option, `INCLUDE_CONVERSIONTAG`, has been added to selectively enable the inclusion of conversion tag information in various mapping probes. This is a bit-encoded value. The bits are assigned are shown in the following table. In all cases the current set of tags appears in the probe as a comma separated list.

Position	Value	Mapping
0	1	`CHARSET_CONVERSION` - added as `;TAG=` field before `;CONVERT`.
1	2	`CONVERSION` - added as `;TAG=` field before `;CONVERT`
2	4	`FORWARD` - added just before current address (| delim)
3	8	`ORIG_SEND_ACCESS` - added at end of probe (| delim)
4	16	`SEND_ACCESS` - added at end of probe (| delim)
5	32	`ORIG_MAIL_ACCESS` - added at end of probe (| delim)
6	64	`MAIL_ACCESS` - added at end of probe (| delim)

To Use Conversion Channel Output Options

Conversion channel output options (Conversion Channel Output Options) are dynamic variables used to pass information and special directives from the conversion script to the conversion channel. For example, during body part processing the script may want to send a special directive asking the conversion channel to bounce the message and to add some error text to the returned message stating that the message contained a virus.

The output options are initiated by setting `OVERRIDE-OPTION-FILE=1` in the desired conversion entry. Output options are then set by the script as needed and stored in the environmental variable file,

OUTPUT_OPTIONS. When the script is finished processing the body part, the conversion channel reads the options from the OUTPUT_OPTIONS file.

The OUTPUT_OPTION variable is the name of the file from which the conversion channel reads options. Typically it is used as an on-the-fly temporary file to pass information. The example below shows a script that uses output options to return an error message to a sender who mailed a virus.

```
/usr/local/bin/viro_screen2k $INPUT_FILE   # run the virus screener

if [ $? -eq 1 ]; then
   echo "OUTPUT_DIAGNOSTIC='Virus found and deleted.'" > $OUTPUT_OPTIONS
   echo "STATUS=178029946" >> $OUTPUT_OPTIONS
else
   cp $INPUT_FILE $OUTPUT_FILE # Message part is OK
fi
```

In this example, the system diagnostic message and status code are added to the file defined by $OUTPUT_OPTIONS. If you read the $OUTPUT_OPTIONS temporary file out you would see something like:

```
OUTPUT_DIAGNOSTIC="Virus found and deleted."
STATUS=178029946
```

The line OUTPUT_DIAGNOSTIC='Virus found and deleted' tells the conversion channel to add the text Virus found and deleted to the message.

178029946 is the PMDF_*FORCERETURN status per the pmdf_err.h file which is found in the _msg-svr-base*/include/deprecated/pmdf_err.h. This status code directs the conversion channel to bounce the message back to the sender. (For more information on using special directives refer to To Bounce, Delete, Hold, Retry Messages Using the Conversion Channel Output.)

A complete list of the output options is shown in the following table.

Conversion Channel Output Options

Option	Description
OUTPUT_TYPE	MIME content type of the output message part.
OUTPUT_SUBTYPE	MIME content subtype of the output message part.
OUTPUT_DESCRIPTION	MIME content description of the output message part.
OUTPUT_DIAGNOSTIC	Text to include as part of the message sent to the sender if a message is forcibly bounced by the conversion channel.
OUTPUT_DISPOSITION	MIME content-disposition of the output message part.
OUTPUT_ENCODING	MIME content transfer encoding to use on the output message part.
OUTPUT_MODE	MIME Mode with which the conversion channel should write the output message part, hence the mode with which recipients should read the output message part.
STATUS	Exit status for the converter. This is typically a special directive initiating some action by the conversion channel. A complete list of directives can be viewed in *msg-svr-base*/include/deprecated/pmdf_err.h

Headers in an Enclosing MESSAGE/RFC822 Part

When performing conversions on a message part, the conversion channel has access to the header in an enclosing MESSAGE/RFC822 part, or to the message header if there is no enclosing MESSAGE/RFC822 part. Information in the header may be useful for the site-supplied program.

If an entry is selected that has ORIGINAL-HEADER-FILE=1, then all the original header lines of the enclosing MESSAGE/RFC822 part are written to the file represented by the ORIGINAL_HEADERS environment variable. If OVERRIDE-HEADER-FILE=1, then the conversion channel will read and use as the header on that enclosing part the contents of the file represented by the ORIGINAL_HEADERS environment variable.

To Call Out to a Mapping Table from a Conversion Entry

out-parameter-* values may be stored and retrieved in an arbitrarily named mapping table. This feature is useful for renaming attachments sent by clients that send all attachments with a generic name like att.dat regardless of whether they are postscript, msword, text, or whatever. This is a generic way to relabel the part so that other clients (Outlook for example) are able to open the part by reading the extension.

The syntax for retrieving a parameter value from a mapping table is as follows:

"mapping-table-name:mapping-input[*$Y*, *$N*]'

$Y returns a parameter value. If there is no match found or the match returns $N, then that parameter in the conversions file entry is ignored or treated as a blank string. Lack of a match or a $N does not cause the conversion entry itself to be aborted.

Consider the following mapping table:

```
X-ATT-NAMES

   postscript       temp.PS$Y
   wordperfect5.1   temp.WPC$Y
   msword           temp.DOC$Y
```

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="cp  "INPUT_FILE'  "OUTPUT_FILE'"
```

In the example above, `out-chan=tcp_local; in-type=application; in-subtype=*` specifies that a message to be processed must come from the `tcp_local` channel with the `content-type` header of `application/*` (* specifies that any subtype would do).

`in-parameter-name-0=name; in parameter-value-0=*` additionally specifies that the message must have a content-type parameter called* `name=*` and that any value for that parameter will be accepted (again, * specifies that any parameter value would do.)

`out-type=application;` specifies that the MIME `Content-type` parameter for the post-processing message be `application`.

`out-subtype='INPUT-SUBTYPE';` specifies that the MIME `subtype` parameter for the post-processing body part be the `INPUT-SUBTYPE` environmental variable, which is the original value of the input `subtype`. Thus, if you wanted change

```
Content-type: application/xxxx; name=foo.doc
```

to

```
Content-type: application/msword; name=foo.doc
```

then you would use

```
out-type=application; out-subtype=msword
```

`out-parameter-name-0=name;` specifies that the output body part will have a MIME Content-type `name=` parameter.

```
out-parameter-value-0='X-ATT-NAMES:\\'INPUT_SUBTYPE\\'';
```

says to take the value of the `INPUT_SUBTYPE` variable (that is, the original content-type header subtype value of the original body part) and search the mapping table `X-ATT-NAMES`. If a match is found, the content-type parameter specified by `out-parameter-name-0` (that is, `name`) receives the new value specified in the `X-ATT-NAMES` mapping table. Thus, if the original subtype was `msword`, the value of the name parameter will be `temp.DOC`.

To Bounce, Delete, Hold, Retry Messages Using the Conversion Channel Output

This section describes how to use the conversion channel options to bounce, delete, or hold messages. The basic procedure is as follows:

1. Set `OVERRIDE-OPTION-FILE=1` in the appropriate conversions file entry. This tells the conversion channel to read the output options from the `OUTPUT_OPTIONS` file.

2. Use the conversion script to determine what action is required on a particular message body part.
3. In the script, specify the special directive for that action by writing the `STATUS=`*directive-code* option in the `OUTPUT_OPTIONS` file.

A complete listing of special directives can be found in *msg-svr-base* `/include/deprecated/pmdf_err.h`. The ones commonly used by the conversion channel are:

Special Directives Commonly Used By the Conversion Channel

NAME	Hex Value	Decimal Value
PMDF__FORCEHOLD	0x0A9C86AA	178030250
PMDF__FORCERETURN	0x0A9C857A	178029946
PMDF__FORCEDELETE	0x0A9C8662	178030178
PMDF__FORCEDISCARD	0x0A9C86B3	178030259
PMDF__AGN	0x0A9C809A	178028698

The functions of these directives will be explained by using examples.

To Bounce Messages

To bounce a message using the conversion channel set `OVERRIDE-OPTION-FILE=1` in the appropriate `conversions` file entry and add the following line to your conversion script:

```
echo "STATUS=178029946" >> $OUTPUT_OPTIONS
```

If you wish to add a short text string to the bounced message add the following line to the conversion script:

`echo OUTPUT_DIAGNOSTIC=`*text-string* `>> $OUTPUT_OPTIONS`

where text string is something like: `"The message sent from your machine contained a virus which has been removed. Be careful about executing email attachments."`

To Conditionally Delete a Message or Its Parts

It may be useful to delete parts conditionally, depending on what they contain. This can be done by using the output options. By contrast, the `DELETE=1` conversion parameter clause unconditionally deletes a message part.

To delete a message part using the output options, set `OVERRIDE-OPTION-FILE=1` in the appropriate conversions file entry and add the following line to your conversion script:

```
echo "STATUS=178030178" >> $OUTPUT_OPTIONS
```

Similarly, to delete the entire message you could use:

```
echo "STATUS=178030259" >> $OUTPUT_OPTIONS
```

To Hold a Message

It may be useful to hold messages conditionally, depending on what they contain. To delete a message part using the output options, set `OVERRIDE-OPTION-FILE=1` in the appropriate conversions file entry and add the following line to your conversion script:

```
echo "STATUS=178030250" >> $OUTPUT_OPTIONS
```

This requests that the conversion channel hold the message as a `.HELD` file in the conversion channel queue.

To Cause Messages to Be Reprocessed

When a converter script encounters a temporary resource problem (for example, the system can't connect to an external server, a needed file is locked, and so on), you can use `PMDF_AGN` to tell the conversion channel to consider processing messages that have encountered a temporary error. The MTA will record a "Q" status message in the `mail.log_current` file, retain the message in the conversion channel, and retry the processing later.

Add the following line to your conversion script:

```
echo "STATUS=178028698" >> $OUTPUT_OPTIONS
```

Conversion Channel Example

The `CONVERSIONS` mapping and set of conversion rules seen in the following examples cause GIF, JPEG, and BITMAP files sent to the hypothetical channel `tcp_docuprint` to be converted into PostScript automatically. Several of these conversions use the hypothetical `/usr/bin/ps-converter.sh` to make that transformation. An additional rule that converts WordPerfect 5.1 files into Microsoft Word files is included.

```
CONVERSIONS

    IN-CHAN=*;OUT-CHAN=tcp_docuprint;CONVERT    Yes
```

```
out-chan=ims-ms; in-type=application; in-subtype=wordperfect5.1;
  out-type=application; out-subtype=msword; out-mode=block;
  command="/bin/doc-convert -in=wp -out=msw   'INPUT_FILE'  'OUTPUT_FILE'"

out-chan=tcp_docuprint; in-type=image; in-subtype=gif;
  out-type=application; out-subtype=postscript; out-mode=text;
  command="/bin/ps-convert -in=gif -out=ps   'INPUT_FILE'  'OUTPUT_FILE'"

out-chan=tcp_docuprint; in-type=image; in-subtype=jpeg;
  out-type=application; out-subtype=postscript; out-mode=text;
  command="/bin/ps-convert -in=jpeg -out=ps  'INPUT_FILE'  'OUTPUT_FILE'"

out-chan=tcp_docuprint; in-type=image; in-subtype=bitmap;
  out-type=application; out-subtype=postscript; out-mode=text;
  command="/bin/ps-convert -in=bmp -out=ps   'INPUT_FILE'  'OUTPUT_FILE'"
```

The conversion parameters are described in the following table:

Conversion Parameters

Parameter	Description

Part 1: Qualifying Parameters (Specifies the parameters for which the message must match before it will be converted.)	
OUT-CHAN,OUT-CHANNEL	Output channel to match for conversion (wildcards allowed). The conversion specified by this entry is performed only if the message is destined for this specified channel.
IN-CHAN,IN-CHANNEL	Input channel to match for conversion (wildcards allowed). The conversion specified by this entry is only performed if the message is coming from the specified channel.
IN-TYPE	Input 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.
IN-SUBTYPE	Input MIME subtype to match for conversion (wildcards allowed). The conversion specified by this entry is performed only if this field matches the MIME subtype of the body part.
IN-PARAMETER-NAME-*n*	Specifies the name of the Input MIME Content-Type parameter that must match for conversion; The n = 0, 1, 2.... is used to optionally pair the specified parameter name requirement with a value required by using IN-PARAMETER-VALUE-n with the same value of *n*.
IN-PARAMETER-VALUE-*n*	Specifies the value required of the input MIME Content-Type parameter whose name is specified in the corresponding IN-PARAMETER-NAME-n. The conversion specified by this entry is performed only if the input body part has the content-type parameter specified by the corresponding IN-PARAMETER-NAME-n and its value matches the value of this parameter. Wildcards allowed.
IN-PARAMETER-DEFAULT-*n*	Default value to use if the input MIME Content-Type parameter specified by the corresponding IN-PARAMETER-NAME-n is not present.
IN-DISPOSITION	Input MIME Content-Disposition to match for conversion.
IN-DPARAMETER-NAME-*n*	Specifies the name of the Input MIME Content-Disposition parameter that must match for conversion; The n = 0, 1, 2.... is used to optionally pair the specified parameter name requirement with a value required by using IN-DPARAMETER-VALUE-n with the same value of *n* .
IN-DPARAMETER-VALUE-*n*	Specifies the value required of the input MIME Content-Disposition parameter whose name is specified in the corresponding IN-DPARAMETER-NAME-n. The conversion specified by this entry is performed only if the input body part has the Content-Disposition parameter specified by the corresponding IN-DPARAMETER-NAME-n and its value matches the value of this parameter. Wildcards allowed.
IN-DPARAMETER-DEFAULT-*n*	Default value to use if the input MIME Content-Disposition parameter specified by the corresponding IN-DPARAMETER-NAME-n is not present.
IN-DESCRIPTION	Input MIME Content-Description to match for conversion.
IN-SUBJECT	Input Subject from enclosing MESSAGE/RFC822 part.

TAG	Input tag, as set by a mail list CONVERSION_TAG parameter.
Part 2: Output Parameters (Specify the body part's post-conversion output settings.)	
OUT-TYPE	Output MIME type if it is different than the input type.
OUT-SUBTYPE	Output MIME subtype if it is different than the input subtype.
OUT-PARAMETER-NAME-*n*	Specifies the name of a content-type parameter which will be set on the output body part.
OUT-PARAMETER-VALUE-*n*	Output MIME Content-Type parameter value corresponding to OUT-PARAMETER-NAME-*n*.
PARAMETER-COPY-*n*	Specifies the name of a content-type parameter which should be copied from the input body part to the output body part.
OUT-DISPOSITION	Output MIME Content-Disposition if it is different than the input MIME Content-Disposition.
OUT-DPARAMETER-NAME-*n*	Output MIME Content-Disposition parameter name; *n*=0, 1, 2...
OUT-DPARAMETER-VALUE-*n*	Output MIME Content-Disposition parameter value corresponding to OUT-DPARAMETER-NAME-*n*.
DPARAMETER-COPY-*n*	A list of the Content-Disposition: parameters to copy from the input body part's Content-Disposition: parameter list to the output body part's Content-Disposition: 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. Wildcards may be used in the argument. In particular, an argument of * means to copy all the original Content-Disposition: parameters.
OUT-DESCRIPTION	Output MIME Content-Description if it is different than the input MIME Content-Description.
OUT-MODE	Mode in which to read and store the converted file. This should be BLOCK (binaries and executables) or TEXT.
OUT-ENCODING	Encoding to apply to the converted file when the message is reassembled.
Part 3: Action Parameters (Specify an action to take on a message part.)	
COMMAND	Command to execute to perform conversion. Command to execute to perform conversion. This parameter is required; if no command is specified, the entry is ignored. Use / to specify paths, not \. Example: command="D:/tmp/mybat.bat"
DELETE	0 or 1. If this flag is set, the message part is deleted. (If this is the only part in a message, then a single empty text part is substituted.)

RELABEL	RELABEL=1 will relabel the MIME label to whatever is specified by the Output parameters. Relabel=0 does nothing. Usually relabelling is done on mislabeled parts (example: from Content-type: application/octet-stream to Content-type: application/msword) so users can "doubleclick" to open a part, rather than having to save the part to a file and open it with a program.
SERVICE-COMMAND	SERVICE-COMMAND=command will execute a site-supplied procedure that will operate on entire MIME message (MIME headers and content body part). Also, unlike other CHARSET-CONVERSION operations or conversion channel operations, the service-command are expected to do their own MIME disassembly, decoding, re-encoding, and reassembly. Note that this flag causes an entry to be ignored during conversion channel processing; SERVICE-COMMAND entries are instead performed during character set conversion processing. Use / to specify paths, not \. Example: command="D:/tmp/mybat.bat"
Part 4: Information Passing Parameters (Used to pass information to and from the site-supplied program.)	
DPARAMETER-SYMBOL-*n*	Environment variable into which the Content-disposition parameter value, if present, will be stored; *n* = 0, 1, 2,... Each DPARAMETER-SYMBOL-*n* is extracted from the Content-Disposition: parameter list in order (n=0 is first parameter, n=2 second, etc.) and placed in the specified environment variable prior to executing the site-supplied program.
PARAMETER-SYMBOL-*n*	Specifies the name of a content-type parameter which, if present in the input body part, its value will be stored in an environment variable of the same name. If the parameter does not exist in the input body part, the environment variable will not exist in the process. For example, if you specify parameter-symbol-0=foo, and there's a content type parameter foo with value bar, you end up with an environment variable foo with value bar.Environment variable into which the Content-Type parameter value, if present, will be stored; *n* = 0, 1, 2... Each PARAMETER-SYMBOL-*n* is extracted from the Content-Type: parameter list in order (n=0 is first parameter, n=2 second, etc.) and placed in an environment variable of the same name prior to executing the site-supplied program. Takes as argument the variable name into which the MIME parameter to convert, as matched by an IN-PARAMETER-NAME-*n* clause.
MESSAGE-HEADER-FILE	If set to 1, the original header of the immediately enclosing body part are written to the file specified by the environmental variable MESSAGE_HEADERS. If set to 2, the original header of the message as a whole (the outermost message header) are written to the file.
ORIGINAL-HEADER-FILE	0 or 1. If set to 1, the original header of the enclosing MESSAGE/RFC822 part (not just the body part) are written to the file represented by the environmental variable ORIGINAL_HEADERS.
OVERRIDE-HEADER-FILE	0 or 1. If set to 1, then MIME header lines are read by the conversion channel from the environmental variable OUTPUT_HEADERS, overriding the original header lines in the enclosing MIME part.
OVERRIDE-OPTION-FILE	If OVERRIDE-OPTION-FILE=1, the conversion channel reads options from the OUTPUT_OPTIONS environmental variable.

PART-NUMBER	Dotted integers: *a. b. c*... The part number of the MIME body part.

Automatic Arabic Character Set Detection

A new `auto_ef` program was added to automatically detect Arabic character sets.

You can call the `auto_ef` program from the conversion channel to automatically detect and label most unlabeled or incorrectly labeled text messages in Arabic character sets. These unlabeled or mislabeled messages are usually sent from Yahoo or Hotmail in Arabic.

Without the correct character set labeling, many mail clients cannot display the messages correctly.

If a message has MIME content-type headers, the `auto_ef` program examines and processes only those with text/plain content type. If the message is not labeled with a MIME content-type header, then `auto_ef` adds a text/plain content-type unconditionally.

To activate or enable this program, you must:

To Automatically Detect Arabic Character Sets

1. Edit your mappings file in the *msg-svr-base*`/config` directory to enable a conversion channel for the source and destination channel of your choosing. To enable a conversion channel for all mail coming in from the Internet to your local users, add a section to your mappings file similar to the following:

   ```
   CONVERSIONS

      IN-CHAN=tcp*;OUT-CHAN=ims-ms;CONVERT YES
   ```

   Note that the `IN` and `OUT` channels depend on your configuration. If you are deploying on a relay MTA, you must modify the channels to fit your configuration. For example,
   `IN-CHAN=tcp*;OUT-CHAN=tcp*;CONVERT YES`
   Or, you could turn it on for all channels as follows:
   `IN-CHAN=`**`;OUT-CHAN=`**`;CONVERT YES`

2. Create a conversions file in the *msg-svr-base*`/config` directory that is owned and readable by the current version of Messaging Server user, and that contains the following:

   ```
   !
   in-channel=*; out-channel=*;
     in-type=text; in-subtype=*;
     parameter-copy-0=*; dparameter-copy-0=*;
     original-header-file=1; override-header-file=1;
     command="_msg-svr-base_
   /lib/arabicdetect.sh"
   !
   ```

3. Compile your MTA configuration with the following command:
   msg-svr-base`/sbin/imsimta cnbuild`
4. Restart with the command:
   msg-svr-base`/sbin/imsimta restart`

Character Set Conversion and Message Reformatting

This section describes character set, formatting, and labelling conversions performed internally by the MTA. Note that some of the examples in this section use old or obsolete technology like DEC VMS, or the d channels. Although these technologies are old or obsolete, this does not make the examples DEC- or d channel-specific. The examples are still valid in describing how the conversion technology works. We will update the examples in a later release.

One very basic mapping table in Messaging Server is the character set conversion table. The name of this table is CHARSET-CONVERSION. It is used to specify what sorts of channel-to-channel character set conversions and message reformatting should be done.

On many systems there is no need to do character set conversions or message reformatting and therefore this table is not needed. Situations arise, however, where character conversions must be done. For example, sites running Japanese OpenVMS may need to convert between DEC Kanji and the ISO-2022 Kanji currently used on the Internet. Another possible use of conversions arises when multinational characters are so heavily used that the slight discrepancies between the DEC Multinational Character Set (DEC-MCS) and the ISO-8859-1 character set specified for use in MIME may become an issue, and actual conversion between the two may therefore be needed.

The CHARSET-CONVERSION mapping table can also be used to alter the format of messages. Facilities are provided to convert a number of non-MIME formats into MIME. Changes to MIME encodings and structure are also possible. These options are used when messages are being relayed to systems that only support MIME or some subset of MIME. And finally, conversion from MIME into non-MIME formats is provided in a small number of cases.

The MTA will probe the CHARSET-CONVERSION mapping table in two different ways. The first probe is used to determine whether or not the MTA should reformat the message and if so, what formatting options should be used. (If no reformatting is specified, the MTA does not bother to check for specific character set conversions.) The input string for this first probe has the general form:

```
IN-CHAN=_in-channel_;OUT-CHAN=_out-channel_;CONVERT
```

Here *in-channel* is the name of the source channel (where the message comes from) and *out-channel* is the name of the destination channel (where the message is going). If a match occurs the resulting string should be a comma-separated list of keywords. The following table lists the keywords.

CHARSET-CONVERSION Mapping Table Keywords

Keyword	Description
`Always`	Force conversion even the message is going to be passed through the conversion channel before going to *out-channel*.
`Appledouble`	Convert other MacMIME formats to Appledouble format.
`Applesingle`	Convert other MacMIME formats to Applesingle format.
`BASE64`	Switch MIME encodings to BASE64. This keyword only applies to message parts that are already encoded. Messages with Content-transfer-encoding: 7BIT or 8bit do not require any special encoding and therefore this BASE64 option will have no effect on them.
`Binhex`	Convert other MacMIME formats, or parts including Macintosh type and Mac creator information, to Binhex format.
`Block`	Extract just the data fork from MacMIME format parts.
`Bottom`	"Flatten" any message/rfc822 body part (forwarded message) into a message content part and a header part.
`Delete`	"Flatten" any message/rfc822 body part (forwarded message) into a message content part, deleting the forwarded headers.
`Level`	Remove redundant multipart levels from message.
`Macbinary`	Convert other MacMIME formats, or parts including Macintosh type and Macintosh creator information, to Macbinary format.
`No`	Disable conversion.
`QUOTED-PRINTABLE`	Switch MIME encodings to QUOTED-PRINTABLE.
`Record,Text`	Line wrap text/plain parts at 80 characters.
`Record,Text= n`	Line wrap text/plain parts at n characters.
`RFC1154`	Convert message to RFC 1154 format.
`Top`	"Flatten" any message/rfc822 body part (forwarded message) into a header part and a message content part.
`UUENCODE`	Switch MIME encodings to X-UUENCODE.
`Yes`	Enable conversion.

Character Set Conversion

If the MTA probes and finds that the message is to be reformatted, it will proceed to check each part of the message. Any text parts are found and their character set parameters are used to generate the second probe. Only when the MTA has checked and found that conversions may be needed does it ever perform the second probe. The input string in this second case looks like this:

```
IN-CHAN=_in-channel_;OUT-CHAN=_out-channel_;IN-CHARSET=_in-char-set_
```

The *in-channel* and *out-channel* are the same as before, and the *in-char-set* is the name of the character set associated with the particular part in question. If no match occurs for this second probe, no character

set conversion is performed (although message reformatting, for example, changes to MIME structure, may be performed in accordance with the keyword matched on the first probe). If a match does occur it should produce a string of the form:

```
OUT-CHARSET=_out-char-set_
```

Here *out-char-set* specifies the name of the character set to which the `in-char-set` should be converted. Note that both of these character sets must be defined in the character set definition table, `charsets.txt`, located in the MTA table directory. No conversion will be done if the character sets are not properly defined in this file. This is not usually a problem since this file defines several hundred character sets; most of the character sets in use today are defined in this file. See the description of the `imsimta chbuild` (UNIX and NT) utility for further information on the `charsets.txt` file.

If all the conditions are met, the MTA will proceed to build the character set mapping and do the conversion. The converted message part will be relabelled with the name of the character set to which it was converted.

The charset-conversion mapping has been extended to provide several additional capabilities:

- A `IN-CHARSET` option can be specified in the output template of a mapping entry. If present this overrides the charset specified in the encoded-word.
- A `RELABEL-ONLY` option that accepts an integer 0 or 1 can be specified. If this option has a value of 1 the `OUT-CHARSET` simply replaces the `IN-CHARSET`; no relabelling is done.
- If the `IN-CHARSET` option is used to set the input charset to * the charset will be "sniffed" to determine an appropriate label.

Converting ISO-8859-1 to UTF-8 and back

Suppose that ISO-8859-1 is used locally, but this needs to be converted to UTF-8 for use on the Internet. In particular, suppose the connection to the Internet is via the `tcp_local` and `tcp_internal` and `ims-ms` are where internal messages originate and are delivered. The `CHARSET-CONVERSION` table shown below brings such conversions about. Note that each `IN-CHAN` entries must be on a single line. The backslash (\) is used to signify this.

```
CHARSET-CONVERSION

 IN-CHAN=tcp_internal;OUT-CHAN=tcp_local;CONVERT               Yes
 IN-CHAN=tcp_local;OUT-CHAN=tcp_internal;CONVERT               Yes
 IN-CHAN=tcp_local;OUT-CHAN=ims-ms;CONVERT                     Yes
 IN-CHAN=*;OUT-CHAN=*;CONVERT                                  No
 IN-CHAN=tcp_internal;OUT-CHAN=tcp_local;IN-CHARSET=ISO-8859-1
OUT-CHARSET=UTF-8
 IN-CHAN=tcp_local;OUT-CHAN=tcp_internal;IN-CHARSET=UTF-8
OUT-CHARSET=ISO-8859-1
 IN-CHAN=tcp_local;OUT-CHAN=ims-ms;IN-CHARSET=UTF-8
OUT-CHARSET=ISO-8859-1
```

Converting EUC-JP to ISO-2022-JP and Back

The `CHARSET-CONVERSION` table shown below specifies a conversion between local usage of EUC-JP and the ISO 2022 based JP code.

```
CHARSET-CONVERSION

  IN-CHAN=ims-ms;OUT-CHAN=ims-ms;CONVERT                    No
  IN-CHAN=tcp_internal;OUT-CHAN=ims-ms;CONVERT              No
  IN-CHAN=tcp_internal;OUT-CHAN=tcp_internal;CONVERT        No
  IN-CHAN=tcp_internal;OUT-CHAN=*;CONVERT                   Yes
  IN-CHAN=*;OUT-CHAN=ims-ms;CONVERT                         Yes
  IN-CHAN=*;OUT-CHAN=tcp_internal;CONVERT                   Yes
  IN-CHAN=tcp_internal;OUT-CHAN=*;IN-CHARSET=EUC-JP
OUT-CHARSET=ISO-2022-JP
  IN-CHAN=*;OUT-CHAN=ims-ms;IN-CHARSET=ISO-2022-JP          OUT-CHARSET=EUC-JP
  IN-CHAN=*;OUT-CHAN=tcp_internal;IN-CHARSET=ISO-2022-JP    OUT-CHARSET=EUC-JP
```

Message Reformatting

As described above, the `CHARSET-CONVERSION` mapping table is also used to effect the conversion of attachments between MIME and several proprietary mail formats.

The following sections give examples of some of the other sorts of message reformatting which can be affected with the `CHARSET-CONVERSION` mapping table.

Non-MIME Binary Attachment Conversion

Mail in certain non-standard (non-MIME) formats; for example, mail in certain proprietary formats or mail from the Microsoft Mail (MSMAIL) SMTP gateway is automatically converted into MIME format if `CHARSET-CONVERSION` is enabled for any of the channels involved in handling the message. If you have a `tcp_local` channel then it is normally the incoming channel for messages from a Microsoft Mail SMTP gateway, and the following will enable the conversion of messages delivered to your local users:

```
CHARSET-CONVERSION

  IN-CHAN=tcp_local;OUT-CHAN=ims-ms;CONVERT         Yes
```

You may also wish to add entries for channels to other local mail systems. For instance, an entry for the `tcp_internal` channel:

```
CHARSET-CONVERSION

  IN-CHAN=tcp_local;OUT-CHAN=l;CONVERT              Yes
  IN-CHAN=tcp_local;OUT-CHAN=tcp_internal;CONVERT   Yes
```

Alternatively, to cover every channel you can simply specify `OUT-CHAN=*` instead of `OUT-CHAN=ims-ms.` However, this may bring about an increase in message processing overhead as all messages coming in the `tcp_local` channel will now be scrutinized instead of just those bound to specific channels.

More importantly, such indiscriminate conversions might place your system in the dubious and frowned upon position of converting messages--not necessarily your own site's--which are merely passing through your system, a situation in which you should merely be acting as a transport and not necessarily altering anything beyond the message envelope and related transport information.

To convert MIME into the format Microsoft Mail SMTP gateway understands, use a separate channel in your MTA configuration for the Microsoft Mail SMTP gateway; for example, `tcp_msmail`, and put the following in the mappings. file:

```
CHARSET-CONVERSION

  IN-CHAN=*;OUT-CHAN=tcp_msmail;CONVERT        RFC1154
```

Relabelling MIME Headers

Some user agents or gateways may emit messages with MIME headers that are less informative than they might be, but that nevertheless contain enough information to construct more precise MIME headers. Although the best solution is to properly configure such user agents or gateways, if they are not under your control, you can instead ask the MTA to try to reconstruct more useful MIME headers.

If the first probe of the `CHARSET-CONVERSION` mapping table yields a `Yes` or `Always` keyword, then the MTA will check for the presence of a `conversions` file. If a conversions file exists, then the MTA will look in it for an entry with `RELABEL=1` and if it finds such an entry, the MTA will then perform any MIME relabelling specified in the entry. See To Control Conversion Processing for information on conversions file entries.

For example, the combination of a `CHARSET-CONVERSION` table such as:

```
CHARSET-CONVERSION

  IN-CHAN=tcp_local;OUT-CHAN=tcp_internal;CONVERT            Yes
```

and MTA `conversion` file entries of

```
out-chan=ims-ms; in-type=application; in-subtype=octet-stream;
  in-parameter-name-0=name; in-parameter-value-0=*.ps;
  out-type=application; out-subtype=postscript;
  parameter-copy-0=*; relabel=1

out-chan=ims-ms; in-type=application; in-subtype=octet-stream;
  in-parameter-name-0=name; in-parameter-value-0=*.msw;
  out-type=application; out-subtype=msword;
     parameter-copy-0=* relabel=1
```

will result in messages that arrive on the `tcp_local` channel and are routed to the `ims-ms` channel, and that arrive originally with MIME labelling of application/octet-stream but have a filename parameter with the extension `ps` or `msw`, being relabelled as application/postscript or application/msword, respectively. (Note that this more precise labelling is what the original user agent or gateway should have performed itself.) Such a relabelling can be particularly useful in conjunction with a `MIME-CONTENT-TYPES-TO-MR` mapping table, used to convert such resulting MIME types back into appropriate `MRTYPE` tags, which needs precise MIME labelling in order to function optimally; if all content types were left labelled only as application/octet-stream, the `MIME-CONTENT-TYPES-TO-MR` mapping table could only, at best, unconditionally convert all such to one sort of `MRTYPE`.

With the above example and `MIME-CONTENT-TYPES-TO-MR` mapping table entries including

```
APPLICATION/POSTSCRIPT        PS
APPLICATION/MSWORD              MW
```

a labelling coming in as, for example,

```
Content-type: application/octet-stream; name=stuff.ps
```

would be relabelled as

```
Content-type: application/postscript
```

and then converted into an MRTYPE tag PS to let Message Router know to expect PostScript.

Sometimes it is useful to do relabelling in the opposite sort of direction, "downgrading" specific MIME attachment labelling to application/octet-stream, the label for generic binary data. In particular, "downgrading" specific MIME labelling is often used in conjunction with the convert_octet_stream channel keyword on the mime_to_x400 channel (PMDF-X400) or xapi_local channel (PMDF-MB400) to force all binary MIME attachments to be converted to X.400 bodypart 14 format.

For instance, the combination of a CHARSET-CONVERSION mapping table such as

```
CHARSET-CONVERSION

    IN-CHAN=*;OUT-CHAN=mime_to_x400*;CONVERT Yes
```

and PMDF conversions file entries of

```
out-chan=mime_to_x400*; in-type=application; in-subtype=*;
   out-type=application; out-subtype=octet-stream; relabel=1

out-chan=mime_to_x400*; in-type=audio; in-subtype=*;
   out-type=application; out-subtype=octet-stream; relabel=1

out-chan=mime_to_x400*; in-type=image; in-subtype=*;
   out-type=application; out-subtype=octet-stream; relabel=1

out-chan=mime_to_x400*; in-type=video; in-subtype=*;
   out-type=application; out-subtype=octet-stream; relabel=1
```

will result in downgrading various specific MIME attachment labelling to the generic application/octet-stream labelling (so that convert_octet_stream will apply) for all messages going to mime_to_x400* channels.

MacMIME Format Conversions

Macintosh files have two parts, a resource fork that contains Macintosh specific information, and a data fork that contains data usable on other platforms. This introduces an additional complexity when transporting Macintosh files, as there are four different formats in common use for transporting the Macintosh file parts. Three of the formats, Applesingle, Binhex, and Macbinary, consist of the Macintosh resource fork and Macintosh data fork encoded together in one piece. The fourth format, Appledouble, is a multipart format with the resource fork and data fork in separate parts. Appledouble is hence the format

most likely to be useful on non-Macintosh platforms, as in this case the resource fork part may be ignored and the data fork part is available for use by non-Macintosh applications. But the other formats may be useful when sending specifically to Macintoshes.

The MTA can convert between these various Macintosh formats. The `CHARSET-CONVERSION` keywords `Appledouble`, `Applesingle`, `Binhex`, or `Macbinary` tell the MTA to convert other MacMIME structured parts to a MIME structure of multipart/appledouble, application/applefile, application/mac-binhex40, or application/macbinary, respectively. Further, the `Binhex` or `Macbinary` keywords also request conversion to the specified format of non-MacMIME format parts that do nevertheless contain `X-MAC-TYPE` and `X-MAC-CREATOR` parameters on the MIME Content-type: header. The `CHARSET-CONVERSION` keyword `Block` tells the MTA to extract just the data fork from MacMIME format parts, discarding the resource fork; (since this loses information, use of `Appledouble` instead is generally preferable).

For example, the following `CHARSET-CONVERSION` table would tell the MTA to convert to Appledouble format when delivering to the VMS MAIL mailbox or a GroupWise postoffice, and to convert to Macbinary format when delivering to the Message Router channel:

```
CHARSET-CONVERSION
   IN-CHAN=*;OUT-CHAN=l;CONVERT              Appledouble
   IN-CHAN=*;OUT-CHAN=wpo_local;CONVERT      Appledouble
   IN-CHAN=*;OUT-CHAN=tcp_internal;CONVERT   Macbinary
```

{{}}

The conversion to Appledouble format would only be applied to parts already in one of the MacMIME formats. The conversion to Macbinary format would only be applied to parts already in one of the MacMIME formats, or non-MacMIME parts which included X-MAC-TYPE and X-MAC-CREATOR parameters on the MIME Content-type: header.

When doing conversion to Appledouble or Block format, the `MAC-TO-MIME-CONTENT-TYPES` mapping table may be used to indicate what specific MIME label to put on the data fork of the Appledouble part, or the Block part, depending on what the Macintosh creator and Macintosh type information in the original Macintosh file were. Probes for this table have the form *format*|*type*|*creator*|*filename* where format is one of SINGLE, BINHEX or MACBINARY, where type and creator are the Macintosh type and Macintosh creator information in hex, respectively, and where filename is the filename.

For example, to convert to Appledouble when sending to the `ims-ms` channel and when doing so to use specific MIME labels for any MS Word or PostScript documents converted from MACBINARY or BINHEX parts, appropriate tables might be:

```
CHARSET-CONVERSION

  IN-CHAN=*;OUT-CHAN=ims-ms;CONVERT     Appledouble


MAC-TO-MIME-CONTENT-TYPES

! PostScript
    MACBINARY|45505346|76677264|*     APPLICATION/POSTSCRIPT$Y
    BINHEX|45505346|76677264|*        APPLICATION/POSTSCRIPT$Y
! Microsoft Word
    MACBINARY|5744424E|4D535744|*     APPLICATION/MSWORD$Y
    BINHEX|5744424E|4D535744|*        APPLICATION/MSWORD$Y
```

Note that the template (right hand side) of the mapping entry must have the $Y flag set in order for the specified labelling to be performed. Sample entries for additional types of attachments may be found in the file `mac_mappings.sample` in the MTA table directory.

If you wish to convert non-MacMIME format parts to Binhex or Macbinary format, such parts need to have X-MAC-TYPE and X-MAC-CREATOR MIME Content-type: parameter values provided. Note that MIME relabelling can be used to force such parameters onto parts that would not otherwise have them.

Service Conversions

The MTA's conversion service facility may be used to process with site-supplied procedures a message so as to produce a new form of the message. Unlike either the sorts of `CHARSET-CONVERSION` operations discussed above or the `conversion` channel, which operate on the content of individual MIME message parts, conversion services operate on entire MIME message parts (MIME headers and content) as well as entire MIME messages. Also, unlike other `CHARSET-CONVERSION` operations or conversion channel operations, conversion services are expected to do their own MIME disassembly, decoding, re-encoding, and reassembly.

Like other `CHARSET-CONVERSION` operations, conversion services are enabled through the `CHARSET-CONVERSION` mapping table. If the first probe of the `CHARSET-CONVESION` mapping table yields a `Yes` or `Always` keyword, then the MTA will check for the presence of an MTA `conversions` file. If a `conversions` file exists, then the MTA will look in it for an entry specifying a `SERVICE-COMMAND`, and if it finds such an entry, execute it. The `conversions` file entries should have the form:

```
in-chan=channel-pattern;
  in-type=type-pattern; in-subtype=subtype-pattern;
  service-command=command
```

Of key interest is the command string. This is the command that should be executed to perform a service conversion (for example, invoke a document converter). The command must process an input file containing the message text to be serviced and produce as output a file containing the new message text. On UNIX, the command must exit with a 0 if successful and a non-zero value otherwise.

For instance, the combination of a `CHARSET-CONVERSION` table such as

```
CHARSET-CONVERSION

IN-CHAN=bsout_;OUT-CHAN=;CONVERT Yes
```

and an MTA `conversions` file entry on UNIX of

```
in-chan=bsout_*; in-type=*; in-subtype=*;
service-command="/pmdf/bin/compress.sh compress $INPUT_FILE $OUTPUT_FILE"
```

will result in all messages coming from a `BSOUT` channel being compressed.

Environment variables are used to pass the names of the input and output files as well as the name of a file containing the list of the message's envelope recipient addresses. The names of these environment variables are:

- `INPUT_FILE` - Name of the input file to process
- `OUTPUT_FILE` - Name of the output file to produce
- `INFO_FILE` - Name of the file containing envelope recipient addresses

The values of these three environment variables may be substituted into the command line by using standard command line substitution: that is, preceding the variable's name with a dollar character on UNIX. For example, when `INPUT_FILE` and `OUTPUT_FILE` have the values `a.in` and `a.out`, then the following declaration on UNIX:

```
in-chan=bsout_*; in-type=*; in-subtype=*;
 service-command="/pmdf/bin/convert.sh $INPUT_FILE $OUTPUT_FILE"
```

executes the command

```
/pmdf/bin/convert.sh a.in a.out
```

Chapter 15. Integrating Spam and Virus Filtering Programs Into Messaging Server

Integrating Spam and Virus Filtering Programs Into Oracle Communications Messaging Server

This information describes how to integrate and configure spam and virus filtering software with Messaging Server. The spam and virus filtering technology is more powerful than the technology provided by the conversion channel (see The Conversion Channel).

Messaging Server supports Symantec Brightmail AntiSpam, SpamAssassin, Milter, and anti-spam/anti-virus programs which support Internet Content Adaptation Protocol (ICAP, RFC 3507), specifically Symantec AntiVirus Scan Engine.

> **Note**
> References to *anti-spam* or *spam filtering* features also mean, when applicable, *anti-virus* or *virus filtering* features. Some products offer both (Brightmail), while others might offer only spam filtering (SpamAssassin) or only virus filtering (Symantec AntiVirus Scan Engine). The term `spam` is used generically in configuration parameters.

Topics:

Integrating Spam Filtering Programs Into Messaging Server - Theory of Operations

From the perspective of Messaging Server, anti-spam solutions operate in much the same way:

1. Messaging Server sends a copy of a message to the spam filtering software.
2. The spam filtering software analyzes the message and returns a verdict of spam or not spam. Some programs, like SpamAssassin may also return a *spam score*, which is a numerical rating of the probability of the message being spam.
3. Messaging Server reads the verdict and takes a Sieve action on the message (see Specifying Actions to Perform on Spam Messages).

Spam filtering programs interact with the MTA through a protocol. The protocol may be a standard as in ICAP-based programs such as Symantec AntiVirus Scan Engine, proprietary as in Brightmail, or simply

non-standard as in SpamAssassin. Each protocol requires software hooks to interface with the MTA. Brightmail and SpamAssassin were the first two spam filtering programs that could be integrated with messaging server. The MTA now supports the programs that use ICAP.

Deploying and Configuring Third Party Spam Filtering Programs

There are five actions required to deploy third-party filtering software on Messaging Server:

1. **Determine which spam filtering programs you wish to deploy, and how many servers on which to deploy them.**
   Messaging Server allows you to filter incoming messages with up to eight different spam/virus programs. These programs can be run on separate systems, on the same system as Messaging Server in a single system deployment, or on the same system as the MTA in a two-tier deployment. The number of servers required depends on the message load, the hardware performance, and other factors. Refer to your spam filtering software documentation or representative for guidelines on determining the hardware requirements at your site.
2. **Install and configure the spam filtering software.** Refer to your spam filtering software documentation or representative for this information.
3. **Load and configure the filtering client library.** This involves specifying the client libraries and configuration files in the MTA `option.dat` file, and also setting the desired options in the filtering software's configuration files. See Loading and Configuring the Spam Filtering Software Client Library.
4. **Specify what messages get filtered.** Messages can be filtered by user, domain, or channel. See Specifying the Messages to Be Filtered.
5. **Specify what happens to Spam.** Spam can be discarded, filed into a folder, tagged on the subject line, and so on. See Specifying Actions to Perform on Spam Messages.

> **Note**
> Previous versions of Messaging Server only supported the Brightmail filtering technology and so keywords and options had names, such as `sourcebrightmail` or `Brightmail_config_file`. These keywords and options have been changed to more generic names such as `sourcespamfilter` or `spamfilter_config_file`. The previous Brightmail names are retained for compatibility.

Loading and Configuring the Spam Filtering Software Client Library

Each spam filtering program is expected to provide a client library file and configuration file for Messaging Server. Loading and configuring the client library involves two things:

- Specifying the spam filtering software library path (`spamfilter`*X*`_library`) and configuration file (`spamfilter`*X*`_config_file`) in the `option.dat` file. In addition to these options, there are a number of others used to specify spam filtering LDAP attributes, as well as the Sieve actions to use on spam messages.
- Specifying the desired options in the spam filtering software configuration files. Each spam filtering program has a different configuration file and configuration options. These are described in the spam filtering software sections, as well as in the filtering software documentation. See Using Symantec Brightmail Anti-Spam and Using Symantec Anti-Virus Scanning Engine (SAVSE).

Specifying the Spam Filtering Software Library Paths

Messaging Server can call up to eight different filtering systems for your messages. For example, you can run your messages through both the Symantec AntiVirus Scan Engine and SpamAssassin. Each filtering software is identified by a number from 1 to 8. These numbers appear as part of the various

spam filter options, LDAP attributes, and channel keywords; an *X* is used as a filter identification number. For example, `sourcespamfilter`*X*`optin` or `spamfilter`*X*`_config_file`. If the identifying number is omitted from the keyword or option name it defaults to 1.

The following `option.dat` settings specify Messaging Server to filter messages through both Symantec AntiVirus Scan Engine and SpamAssassin:

```
spamfilter1_library=<Symantec_Library_File>
spamfilter1_config_file=<Symantec_Config_File>
spamfilter2_library=<SpamAssassin_Library_File>
spamfilter2_config_file=<SpamAssassin_Config_File>
```

When using other options or keywords to configure the system, use the corresponding number at the end of the option or keyword. For example, `sourcespamfilter2optin` would refer to SpamAssassin. `sourcespamfilter1optin` would refer to Symantec AntiVirus Scan Engine. It is not necessary to use numbers sequentially. For example, if you want to temporarily disable the Symantec AntiVirus Scan Engine, you can just comment out the `spamfilter1_library` configuration file.

Specifying the Messages to Be Filtered

Once the spam filtering software is installed and ready to run with Messaging Server, you need to specify what messages to filter. Messaging Server can be configured to filter messages by user, domain, or channel. Each of these scenarios is described in the following sections:

- Integrating Spam Filtering Programs Into Messaging Server - Theory of Operations
- Deploying and Configuring Third Party Spam Filtering Programs
- Using Symantec Brightmail Anti-Spam
- Using SpamAssassin
- Using Symantec Anti-Virus Scanning Engine (SAVSE)
- Using ClamAV
- Support for Sieve Extensions
- Using Milter
- Cloudmark Anti-Abuse Client
- Other Anti-Spam and Denial-of-Service Technologies

Note
The expression *optin* means that a user, domain or channel is selected to receive mail filtering.

To Specify User-level Filtering

It may be desirable to specify filtering on a per-user basis. For example, if spam or virus filtering is offered as a premium service to ISP customers, you can specify which users receive this and which don't. The general steps for user filtering are as follows:

1. Specify the user LDAP attributes that activate the spam filtering software.
   Set the `LDAP_OPTIN`*X* options in `option.dat`. Example:

   ```
   LDAP_OPTIN1=SymantecAV
   LDAP_OPTIN2=SpamAssassin
   ```

> **Note**
> By default, the attributes like `SymantecAV` or `SpamAssassin` do not exist in the schema. Whatever new attributes you use, you will need to add them to your directory schema. See the appropriate Directory Server documentation for instructions.

2. Set filter attributes in the user entries that receive spam filtering.
   The values for the filter attributes are multi-valued and depend on the server. Using the example shown in Step 1, the entries are:

```
SymantecAV: virus
SpamAssassin: spam
```

   For a program like Brightmail, which can filter both viruses and spam, the valid values are `spam` and `virus`. When used as a multi-valued attribute, each value requires a separate attribute entry. For example, if the filter attribute for Brightmail was set to `Brightmail`, the entries are:

```
Brightmail: spam
Brightmail: virus
```

User-level Filtering Example

This example assumes that Brightmail is used. It also assumes that `LDAP_OPTIN1` was set to Brightmail in the `option.dat` file. The user, Otis Fanning, has the Brightmail attribute set to `spam` and `virus` in his user entry. His mail is filtered by Brightmail for spam and viruses. User-level Filtering Example shows the Brightmail user entry for Otis Fanning.

Example LDAP User Entry for Brightmail

```
dn: uid=fanning,ou=people,o=sesta.com,o=ISP
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: inetUser
objectClass: ipUser
objectClass: inetMailUser
objectClass: inetLocalMailRecipient
objectClass: nsManagedPerson
objectClass: userPresenceProfile
cn: Otis Fanning
sn: fanning
initials: OTF
givenName: Otis
pabURI:
ldap://ldap.siroe.com:389/ou=fanning,ou=people,o=sesta.com,o=isp,o=pab
mail: Otis.Fanning@sesta.com
mailAlternateAddress: ofanning@sesta.com
mailDeliveryOption: mailbox
mailHost: manatee.siroe.com
uid: fanning
dataSource: iMS 5.0 @(#)ims50users.sh 1.5a 02/3/00
userPassword: password
inetUserStatus: active
mailUserStatus: active
mailQuota: -1
mailMsgQuota: 100
Brightmail: virus
Brightmail: spam
```

If Symantec AntiVirus Scan Engine and SpamAssassin were used, the entry would look like this:

```
SymantecAV: virus
SpamAssassin: spam
```

See Using Symantec Brightmail Anti-Spam, Using SpamAssassin or Using Symantec Anti-Virus Scanning Engine (SAVSE).

To Specify Domain-level Filtering

You can specify which domains receive filtering. An example of this feature would be if anti-spam or anti-virus filtering were offered as a premium service to ISP domain customers. The general steps for specifying domain filtering is as follows:

1. Specify the domain LDAP attributes that activates the filtering software.
   Set the `LDAP_DOMAIN_ATTR_OPTIN`*X* options in `option.dat`. Example:

   ```
   LDAP_DOMAIN_ATTR_OPTIN1=SymantecAV
   LDAP_DOMAIN_ATTR_OPTIN2=SpamAssassin
   ```

> **Note**
> By default, the attributes like `SymantecAV` or `SpamAssassin` do not exist in the schema. Whatever new attributes you use, you will need to add them to your directory schema. See the appropriate Directory Server documentation for instructions.

2. Set filter attributes in the domain entries that receive spam filtering.
   The values for the filter attributes are multi-valued and depend on the server. Using the example shown in Step 1, the entries would be as follows:

```
SymantecAV: virus
SpamAssassin: spam
```

   For a program like Brightmail which can filter both viruses and spam, the valid values are `spam` and `virus`. When used as a multi-valued attribute, each value requires a separate attribute value entry. For example, if `LDAP_DOMAIN_ATTR_OPTIN1` was set to `Brightmail`, the entries would be:

```
Brightmail: spam
Brightmail: virus
```

Domain-level Filtering Example

This example assumes that Brightmail is used. It also assumes that `LDAP_DOMAIN_ATTR_OPTIN1` was set to `Brightmail` in the `option.dat` file. The Brightmail attribute is set to `spam` and `virus` in the `sesta.com` domain entry in the DC tree for Sun LDAP Schema 1. For Sun LDAP Schema 2 you also set `Brightmail` in the domain entries that receive spam filtering.

All mail sent to `sesta.com` is filtered for spam and viruses by Brightmail. A Domain-level Filtering Example is shown below.

Example LDAP Domain Entry for Brightmail

```
dn: dc=sesta,dc=com,o=internet
objectClass: domain
objectClass: inetDomain
objectClass: mailDomain
objectClass: nsManagedDomain
objectClass: icsCalendarDomain
description: DC node for sesta.com hosted domain
dc: sesta
inetDomainBaseDN: o=sesta.com,o=isp
inetDomainStatus: active
mailDomainStatus: active
mailDomainAllowedServiceAccess: +imap, pop3, http:*
mailRoutingHosts: manatee.siroe.com
preferredMailHost: manatee.siroe.com
mailDomainDiskQuota: 100000000
mailDomainMsgQuota: -1
mailClientAttachmentQuota: 5
Brightmail: spam
Brightmail: virus
```

If Symantec AntiVirus Scan Engine and SpamAssassin were used, the entry would look similar to like this:

```
SymantecAV: virus
SpamAssassin: spam
```

See Using Symantec Brightmail Anti-Spam, Using SpamAssassin, or Using Symantec Anti-Virus Scanning Engine (SAVSE) for more examples and details.

To Specify Channel-level Filtering

Filtering by source or destination channel provides greater flexibility and granularity for spam filtering. For example, you may wish to filter in these ways:

- Only messages from a specific MTA relay to a backend message store
- All incoming mail from a specific MTA.
- All outgoing mail from a specific MTA.
- Incoming and outgoing mail from a specific MTA.

Messaging Server allows you to specify filtering by source or destination channel. The mechanism for doing this are the channel keywords described in Configuring Channel Definitions. The following example demonstrates how to set up channel-level filtering.

1. Add a rewrite rule in the `imta.cnf` file for all inbound SMTP servers that send messages to a backend message store host. Example:
   `msg_store1.siroe.com $U@msg_store1.siroe.com`
2. Add a channel corresponding to the rewrite rule with the `destinationspamfilter`*X*`optin` keyword. Example:

   ```
   tcp_msg_store1 smtp subdirs 20 backoff "pt5m" "pt10" "pt30" \
   "pt1h" "pt2h" "pt4h" maxjobs 1 pool IMS_POOL \
   fileinto $U+$S@$D destinationspamfilter1optin spam
   msg_store1.siroe.com
   ```

Channel-level Filtering Examples

These examples assume a filtering program specified by the number 1. See Spam Filter Keywords for the keywords available for spam filtering.

Example 1. To Filter from an MTA Relay to a Backend Message Store
This example filters all mail for spam and viruses from an MTA relay to a backend message store called `msg_store1.siroe.com`.

1. Add a rewrite rule in the `imta.cnf` file that sends messages to a backend message store host. Example:

```
msg_store1.siroe.com $U@msg_store1.siroe.com
```

2. Add a channel corresponding to that rewrite rule with the `destinationspamfilter`*X*`optin` keyword. Example:

```
tcp_msg_store1 smtp subdirs 20 backoff "pt5m" "pt10" "pt30" "pt1h"
\
"pt2h" "pt4h" maxjobs 1 pool IMS_POOL fileinto $U+$S@$D \
destinationspamfilter1optin spam,virus
msg_store1.siroe.com
```

Example 2. Filter for spam all incoming mail passing through your MTA with the `sourcespamfilter`*X* `optin` keyword. (Typically, all incoming messages pass through the `tcp_local` channel):

```
tcp_local smtp mx single_sys remotehost inner switchchannel \
identnonelimited subdirs 20 maxjobs 7 pool SMTP_POOL \
maytlsserver maysaslserver saslswitchchannel tcp_auth \
sourcespamfilter1optin spam
tcp-daemon
```

Example 3. Filter all outgoing mail to the Internet passing through your MTA. (Typically, all messages going out to the Internet pass through the `tcp_local` channel.)

```
tcp_local smtp mx single_sys remotehost inner switchchannel \
identnonelimited subdirs 20 maxjobs 7 pool SMTP_POOL \
maytlsserver maysaslserver saslswitchchannel tcp_auth \
destinationspamfilter1optin spam
tcp-daemon
```

Example 4. Filter all incoming and outgoing mail passing through your MTA:

```
tcp_local smtp mx single_sys remotehost inner switchchannel \
identnonelimited subdirs 20 maxjobs 7 pool SMTP_POOL \
maytlsserver maysaslserver saslswitchchannel tcp_auth \
sourcespamfilter1optin spam destinationspamfilter1optin spam
tcp-daemon
```

Example 5. Filter all mail destined to the local message store in a two-tiered system without using user optin:

```
ims-ms smtp mx single_sys remotehost inner switchchannel \
identnonelimited subdirs 20 maxjobs 7 pool SMTP_POOL \
maytlsserver maysaslserver saslswitchchannel tcp_auth \
destinationspamfilter1optin spam
ims-ms-daemon
```

Example 6. Filter all incoming and outgoing mail for spam and viruses (this presumes that your software filters both spam and viruses):

```
tcp_local smtp mx single_sys remotehost inner switchchannel \
identnonelimited subdirs 20 maxjobs 7 pool SMTP_POOL \
maytlsserver maysaslserver saslswitchchannel tcp_auth \
destinationspamfilter1optin spam,virus \
sourcespamfilter1optin spam,virus
tcp-daemon
```

Specifying Actions to Perform on Spam Messages

Spam filtering programs analyze messages and return a verdict of spam or not spam to current version of Messaging Server. Messaging Server then takes action on the message. Actions are specified using the Sieve mail filtering language. Possible actions are to discard the message, file it into a folder, add a header, add a tag to the subject line, and so on. Complex Sieve scripts with if-then-else statements are also possible.

> **Note**
> Refer to the Sieve specification RFC 3028 for the complete Sieve syntax.

Sieve scripts are specified with the MTA spam filter options (`option.dat`) described in Table. The primary spam filter action options are `SpamfilterX_null_action`, which specifies the Sieve rule to execute when a null value is returned as the spam verdict value, and `SpamfilterX_string_action`, which specifies the Sieve rule to execute when a string is returned as the spam verdict.

Spam filtering programs typically return a string or a null value to the MTA to indicate that message is spam. Some programs also return a spam score---a number rating the probability of the message being spam or not. This score can be used as part of the action sequence. The following examples show how to specify actions on filtered messages. Each example assumes a filtering program specified by the number 1.

Example 1: File spam messages with a null verdict value in the mailbox `SPAM_CAN`.

```
spamfilter1_null_action=data:,require ["fileinto"]; fileinto "SPAM_CAN";
```

The same action can be performed on a spam message that returns a string:

```
spamfilter1_string_action=data:,require ["fileinto"]; fileinto
"SPAM_CAN";
```

Example 2: File spam messages with a returned verdict string in the mailbox named after the returned verdict string (this is what `$U` does). That is, if the verdict string returned is `spam`, the message is stored in a folder called `spam`.

```
spamfilter1_null_action=data:,require ["fileinto"]; fileinto "$U";
```

Example 3: Discard spam messages with a string verdict value.

```
spamfilter1_string_action=data:,discard;
```

The same action can be performed on a spam message that returns a null value:

```
spamfilter1_null_action=data:,discard;
```

Example 4. This line adds the header `Spam-test: FAIL` to each message determined to be spam by a string verdict value:

```
spamfilter1_string_action=data:,require ["editheader"];addheader
"Spam-test" "FAIL";
```

Example 5. This line adds the string [PROBABLE SPAM] to the subject line of the spam messages returning a string:

```
spamfilter1_string_action=data:,addtag '[PROBABLE SPAM]';
```

Example 6. This line assumes a string verdict value and files a spam message in the mailbox `testspam` if the header contains `resent-from` and `User-1`. If the message does not have that header, it files the message into `spam.`

```
spamfilter1_string_action=data:,require ["fileinto"]; \
if header :contains ["resent-from"] ["User-1"] { \
fileinto "testspam"; \
} else { \
fileinto "spam";};
```

Because verdict strings are configurable with most spam filter software, you can specify different actions depending on the returned string. This can be done with the matched pairs `spamfilter`*X*`_verdict_`*n* and `spamfilter`*X*`_action_`*n* options.

Example 7. These matched pair options discard spam messages with the returned verdict string of `remove`.

```
spamfilter1_verdict_0=remove
spamfilter1_action_0=data:,discard;
```

Refer to the specific spam filtering software sections for instructions on how to specify the spam verdict string.

MTA Spam Filter Options (option.dat)

MTA Options	Description

spamfilterX_ config_file	Specifies the full file path and name of the filtering software X configuration file. The value of this option is passed to the filtering software for it to use to locate its configuration file. Whether the filtering software prefers absolute file paths (including full directory path), or prefers a bare file name (presumably located in some fixed/default directory), can vary with the specific filtering package in use. Default: none
spamfilterX_ library	Specifies the full file path and name of the filtering software X shared library. In this way, the MTA knows where to find the filtering software code and use it. Full MTA file name/file reading handling is always available for these options. Default: none
spamfilterX_ optional	Controls whether certain failures reported by the filtering library X are treated as a temporary processing failure or ignored. 0 specifies that spam filtering problems cause a temporary processing failure. 1 causes spam filter processing to be skipped in the event of some, but possibly not all, filtering library failures. In particular, if the system gets stuck without a return in the library code, some portion of the MTA may also get stuck. -2 and 2 are the same as 0 and 1 respectively except that they also cause a syslog message to be sent in the event of a problem reported by the spam filter plugin. 3 causes spam filter failures to accept the message, but queue it to the reprocess channel for later processing. 4 does the same as 3 but also logs the spam filter temporary failure to syslog. Default: 0
LDAP_optinX	Specifies the name of the LDAP attribute used to activate filtering software X on a per-user basis. Filtering is based on destination addresses. That is, messages directed to users with this attribute will be filtered for spam. This should be an attribute in the inetMailUser objectclass.The attribute itself can take multiple values and is case-sensitive. For SpamAssassin, its value should be spam in lowercase. Default: none
LDAP_ SOURCE_ OPTINX	LDAP_SOURCE_OPTIN1 through LDAP_SOURCE_OPTIN8 provide originator-address-based per-user spam filter optins values comparable to LDAP_optinX. This is, mail originating from this user will be filtered for spam.
LDAP_domain_ attr_optinX	Specifies the name of the LDAP attribute used to activate filtering software X on a domain basis. It applies to the destination domain. It is just like LDAP_optin, except it should be in the objectclass mailDomain. Default: none
spamfilterX_ null_optin	Specifies a string which, if found, as a value of the attribute defined by LDAP_optinX or LDAP_domain_attr_optinX, causes the MTA to act as if the attribute wasn't there. That is, it disables filtering for that entry. See Specifying the Messages to Be Filtered. Default: The empty string. Empty optin attributes are ignored by default. (This is a change from iPlanet Messaging Server 5.2, where empty optin attributes triggered filtering with an empty optin list. The 5.2 behavior can be restored by setting spamfilterX_null_optin to a string that never occurs in practice.)

spamfilterX_ null_action	Defines a Sieve rule specifying what to do with the message when the filtering software X verdict returns as null. Sieve expressions can be stored externally using a file URL. For example: file:///opt/sun/comms/messaging/config/null_action.sieve. Also, do not reject spam using the Sieve reject action, as it tends to deliver a nondelivery notification to the innocent party whose address was used to send the spam. Default: data:,discard;
spamfilterX_ string_action	Defines Sieve rule specifying what to do with the message if the verdict is a string. Sieve expressions can be stored externally using a file URL. For example: file:///opt/sun/comms/messaging/config/null_action.sieve. Also, do not reject spam using the Sieve reject action, as it tends to deliver a nondelivery notification to the innocent party whose server was used to send the spam. Default: data:,require "fileinto"; fileinto "$U; where $U is the string that verdict returned.
spamfilterX_ verdict_n	The options spamfilterX_verdict_n and spamfilterX_action_n are matched pairs, where *n* is a number from 0 to 9. These options allow you to specify Sieve filters for arbitrary verdict strings. This is done by setting spamfilterX_verdict_n and spamfilterX_action_n to the verdict string and sieve filter, respectively, where *n* is an integer from 0 to 9. For example, a site could have the "reject"; verdict cause a sieve reject action by specifying: spamfilter1_verdict_0=reject spamfilter1_action_0=data:,require "reject"; reject "Rejected by spam filter"; The default values for all the spamfilterX_verdict_n options and the corresponding action options are empty strings. Default: none
spamfilterX_ action_n	See spamfilterX_verdict_n. Default: none
spamfilterX_ final	Some filtering libraries have the ability to perform a set of actions based on recipient addresses. What sort of recipient address is passed to the filtering library depends on the setting of the spamfilterX_final. The default value of 0 results in a so-called intermediate address being passed to the filtering library. This address is suitable for use in delivery status notifications and for directory lookups. If bit 0 (value 1) of spamfilter{}*X*_final is set, however, the final form of the recipient address is passed. This form may not be suitable for presentation but is more appropriate for subsequent forwarding operations.The spamfilterX_final option is only available in Messaging Server 6.0 and above; Messaging Server 5.2 behaves as if the option had the default value of 0. In Messaging Server 6.2 and later bit 1 (value 2) of spamfilterX_final controls whether or not source routes are stripped from the address that's passed to the filtering interface. Setting the bit enables source route stripping. Default: 0
spamfilterX_ returnpath	(New in Messaging Server 7 Update 1) If set to 1 this will cause a Return-path: header to be added to the message passed to the corresponding filter. 0 disables this addition. Default: 0

optin_user_carryover	Forwarding is a challenge for spam filter processing. Consider a user entry that specifies the forward delivery option and specifies the forwarding address of another user. Additionally, the user entry is set to opt in to some specific sort of filtering. Should the filtering be applied to the forwarded message or not? On the one hand, the correct filtering choice for one particular user may not be the correct choice for another. On the other hand, eliminating a filtering operation might be used as means of violating a site's security policy. No single answer is correct in all cases so OPTIN_USER_CARRYOVER controls how the spam filtering optin list is carried from one user or alias entry to another when forwarding occurs. This is a bit-encoded value. The various bit values have the following meanings: bit 0 (value 1). Each LDAP user entry overrides any previously active user/domain optins unconditionally. bit 1 (value 2). If a user's domain has an optin attribute, it overrides any previous user/domain/alias optins that were active. bit 2 (value 4). If a user has an optin attribute, it overrides any previous user/domain/alias optins that were active. bit 3 (value 8). An optin specified by an [optin] non-positional parameter overrides any previous user/domain/alias optins that were active. Default: 0 (optins accumulate if one user has a delivery option that forwards to another. The default insures that site security policies are effective when forwarding; other settings may not.)

Using Symantec Brightmail Anti-Spam

The Brightmail solution consists of the Brightmail server along with realtime anti-spam and anti-virus rule updates downloaded to email servers. In addition to the sections below, refer to Configuring Brightmail with Sun Java System Messaging Server.

- How Brightmail Works
- Brightmail Requirements and Performance Considerations
- Deploying Brightmail
- Brightmail Configuration Options

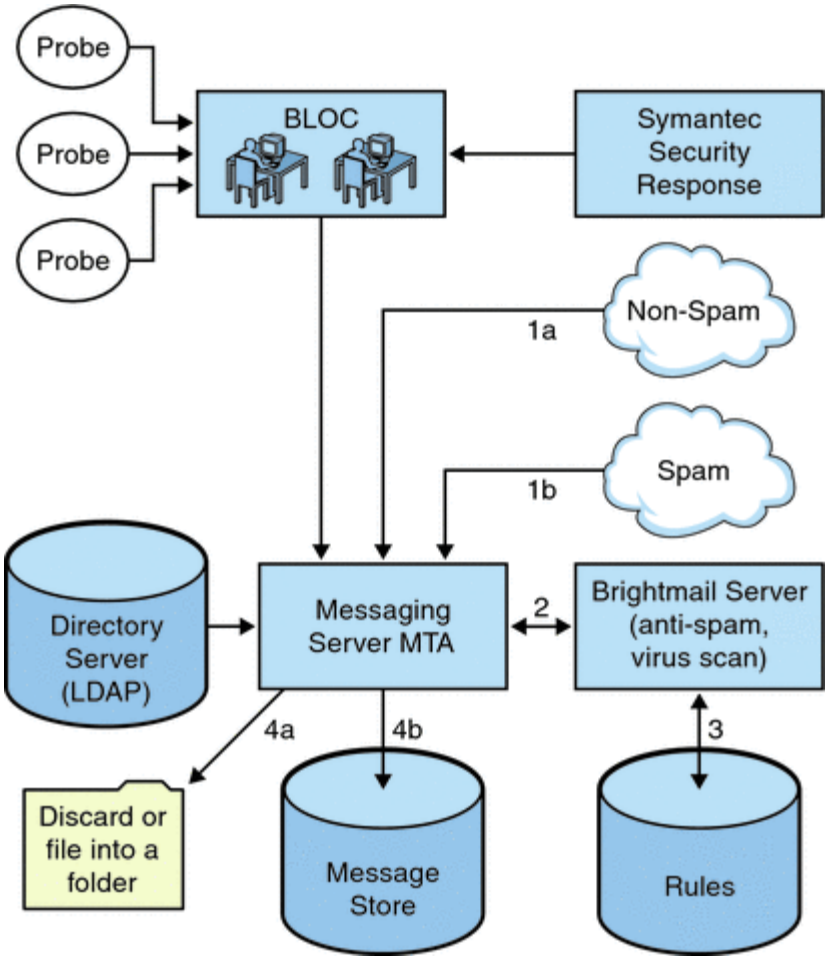
How Brightmail Works

The Brightmail server is deployed at a customer site. Brightmail has email probes set around the Internet for detection of new spam. Brightmail technicians create custom rules to block this spam in realtime. These rules are downloaded to Brightmail servers, also in realtime. The Brightmail database is updated and Brightmail server runs this database filter against the email for the specified users or domains.

Brightmail Architecture

The following figure depicts the Brightmail architecture.

Brightmail and Messaging Server Architecture



When the Brightmail Logistics and Operations Center (BLOC) receives spam from email probes, operators immediately create appropriate spam filtering rules, which are downloaded to Brightmail customer machines. Similarly, the Symantec Security Response realtime virus rules are also sent from Brightmail. These rules are used by customer's Brightmail servers to catch spam and viruses.

The MTA uses the Brightmail SDK to communicate with the Brightmail Server. The MTA dispatches messages based on the response from Brightmail. After the mail (1a) or (1b) is received by the MTA, the MTA sends the message to the Brightmail server (2). The Brightmail server uses its rules and data to determine if the message is a spam or virus (3), and returns a verdict to the MTA. Based on the verdict, the MTA either (4a) discards the message or files the message into a folder, or (4b) delivers it normally to the destination.

Since the Brightmail SDK is third party software, we do not include it in our installation kit. The Brightmail SDK and server software must be obtained from Brightmail Inc. The MTA has configuration settings to tell it whether and where to load the Brightmail SDK to enable Brightmail integration.

Once the SDK is loaded, Brightmail message processing is determined by several factors and levels of granularity (the term used by Brightmail to specify active processing is *optin*). This is specified by the following criteria:

- Whether the source or destination channel is enabled for Brightmail (`imta.cnf`)
- Whether there is a channel default for the services opted in (`imta.cnf`)
- Whether there is a per domain optin (LDAP)
- Whether there is per-user optin (LDAP)

For any particular message recipient, the optins and defaults above are combined, which means, if the channel default is already specified for both spam and virus, then there is no reason to bother with per-user optin. That is, if the system administrator decides to do spam and virus filtering for everyone,

then there is no reason to expose to the user the ability to optin for spam or virus. There is no way to opt out of processing, that is, you can not say you do not want the service if a user is already optin via a system or domain optin. This also means that if you are optin for a service, and you have forwarded your mail to another address, that address would get the mail after the filtering has been performed on your behalf.

There are only two services offered, virus or spam detection. Brightmail also provides "content-filtering" service, but this functionality is provided using Sieve, so there is no added value to have Brightmail do the Sieve filtering.

When a message is determined to contain a virus, the Brightmail server can be configured to clean the virus and resubmit the cleaned message back to the MTA. (Due to some undesirable side effects caused by loss of information about the original message in a resubmitted cleaned message, we recommend you do not configure Brightmail to resubmit the cleaned message back to the MTA.) When the message is spam, the verdict back from the Brightmail along with the configuration in Brightmail allows the MTA to determine what happens to the message. The message can be discarded, filed into a folder, tagged as spam or virus on the subject line, passed to a Sieve rule, delivered normally in the `INBOX`, and so on.

The Brightmail servers can be located on the same system as the MTA, or it can be on a separate system. In fact, you can have a farm of Brightmail servers serving one or more MTAs. The Brightmail SDK uses the Brightmail configuration file to determine which Brightmail servers to use.

Brightmail Requirements and Performance Considerations

- Brightmail servers must run on Oracle Solaris.
- If Brightmail does spam and virus checking, MTA message throughput can be reduced by as much 50%. To keep up with MTA throughput, you may need two Brightmail servers for each MTA.
- Whilst SpamAssassin has the ability to perform different sorts of filtering on a user basis, it is unable to apply two different sets of filtering criteria to the same message at the same time. Thus, SpamAssassin only allows system-wide filtering. Customized filtering for individual users is not possible.

Deploying Brightmail

Perform the following steps to deploy Brightmail.

- **Install and Configure Brightmail.** Refer to the Brightmail software documentation or representative for installation and configuration information. Selected Brightmail configuration options are shown in Brightmail Configuration Options, however the most complete and up-to-date information is in the Brightmail documentation.
- **Load and Configure the Brightmail Client Library.** This involves specifying the Brightmail client library, `libbmiclient.so`, and configuration, `config`, file to the MTA. See Loading and Configuring the Spam Filtering Software Client Library.
- **Specify what messages to filter for spam.** Messages can be filtered by user, domain, or channel. See Specifying the Messages to Be Filtered.
- **Specify what actions to take on spam messages.** Spam can be discarded, filed into a folder, tagged on the subject line, and so on. See Specifying Actions to Perform on Spam Messages.
- **Set miscellaneous MTA filter configuration parameters as desired.** See Table.

Brightmail Configuration Options

Selected Brightmail configuration file options are shown in Table. The most complete listing of Brightmail configuration file options can be obtained from Brightmail. Options and values are not case-sensitive.

Selected Brightmail Configuration File Options

Brightmail Option	Description

`blSWPrecedence`	A given message can have multiple verdicts. This specifies the precedence order. So if a message is processed for virus first, then for spam if you specified this option as `virus-spam` the verdicts are separated by hyphens (-). This is the recommended setting when using Brightmail with Sun Java System Messaging Server.
`blSWClientDestinationDefault`	Specifies how to deliver normal messages, that is, not a spam or virus, and thus have no verdict. Usually you want to deliver this message normally, so you would specify `inbox` as the value. There is no default.
`blSWLocalDomain`	This attribute specifies what domain(s) are considered to be local. There can be multiple lines of this attribute specifying several domains which are all considered local. Local versus foreign domain is used to specify two different handling for a verdict. See below `blSWClientDestinationLocal` and `blSWClientDestinationForeign`. For example, you can specify `blSWLocalDomain=siroe.com`

`blSWClientDestinationLocal`	This specifies the verdict and action pair for the local domain. You would normally have two lines for this, one for spam and one for virus. The value is of the form `verdict|action`, For example,
	
	`blSWClientDestinationLocal=spam|spambox`
	`blSWClientDestinationLocal=virus|`
	
	The default Brightmail interpretation for the "null" action, meaning nothing to the right of the |, is to discard the message. So the example above discards the message if it has a verdict of `virus`. And if the verdict is `spam`, the above example files the message into the folder called `spambox`. If the message is not spam or virus, then the verdicts do not match, and the mail is delivered normally based on what's set in the `blSWClientDestinationDefault` setting above.
	
	When using a separate Brightmail server or servers from the MTA, you can customize the actions taken by each MTA by using the `Brightmail_verdict_n`, `Brightmail_action_n`, `Brightmail_null_action`, and `Brightmail_string_action` MTA options to override the actions and verdicts returned by the Brightmail server. In this example, you can use different `Brightmail_null_action` on the MTA to override the Virus action (which would be to discard it) or to use `Brightmail_verdict_0=spambox`, and `Brightmail_action_0=data:,require "fileinto";fileinto "Junk";` to file into a folder called `Junk` instead of `spambox`.

`blSWClientDesintationForeign`	Same format and interpretation as `blSWClientDestinationLocal` above, except this applies to users in the domain which are NOT local.
`blSWUseClientOptin`	Always set this to TRUE when used with Sun Java System Messaging Server.
`blswcServerAddress`	Is of the form `ip:port[,ip:port,...]` to specify one or more Brightmail server's IP address and port numbers

Using SpamAssassin

This section consists of the following subsections:

SpamAssassin Overview

Messaging Server supports the use of SpamAssassin, a freeware mail filter used to identify spam. SpamAssassin consists of a library written in Perl and a set of applications and utilities that can be used to integrate SpamAssassin into messaging systems.

SpamAssassin calculates a score for every message by performing a series of tests on the message header and body information. Each test succeeds or fails, and a verdict of true (spam) or false (not spam) is rendered. Scores are real numbers that can be positive or negative. Scores that exceed a specified threshold, typically 5.0, are considered to be spam. An example of a SpamAssassin result string is:

True ; 18.3 / 5.0

True indicates the message is spam. *18.3* is the SpamAssassin score. *5.0* is the threshold.

SpamAssassin is highly configurable. Tests may be added or removed at any time, and the scores of existing tests can be adjusted. This is all done through various configuration files. Further information on SpamAssassin can be found on the SpamAssassin web site.

The same mechanism used for calling out to the Brightmail spam and virus scanning library can be used to connect to the SpamAssassin `spamd` server. The module provided in Messaging Server is called `libspamass.so`.

SpamAssassin/Messaging Server Theory of Operations

`spamd` is the daemon version of SpamAssassin and can be invoked from the MTA. `spamd` listens on a socket for requests and spawns a child process to test the message. The child process dies after processing the message and sending back a result. In theory, the forking should be an efficient process because the code itself is shared among the children processes.

The client portion, `spamc` from the SpamAssassin installation, is not used. Instead, its function is done by a shared library called `libspamass.so`, which is part of the Messaging Server. `libspamass.so` is loaded the same way that the Brightmail SDK is loaded.

From the MTA's point of view, you can almost transparently switch between SpamAssassin and Brightmail for spam filtering. It's not completely transparent, because they do not have the same functions. For example, Brightmail can also filter for viruses, but SpamAssassin is only used to filter for spam. The result, or *verdict*, returned by the two software packages is also different. SpamAssassin provides a score, while Brightmail provides just the verdict name, so the configuration would also have some differences.

When using SpamAssassin with the MTA, only a score and verdict is returned from SpamAssassin. The message itself is not modified. That is, options such as adding headers and modifying subject lines must be done by Sieve scripts. In addition, the `mode` option allows you to specify the string that is returned to indicate the verdict. The string choices are null, default, SpamAssassin result string, or a `verdict` string. See SpamAssassin Options for details.

SpamAssassin Requirements and Usage Considerations

- SpamAssassin is free. Go to http://www.spamassassin.org for software and documentation.
- SpamAssassin can be tuned and configured to provide very accurate detection of spam. The tuning is up to you and the SpamAssassin community. Messaging Server does not provide or enhance what SpamAssassin can do.
- While no specific numbers are available, SpamAssassin seems to reduce throughput more than Brightmail.
- SpamAssassin integrated with the MTA can be enabled for a user, a domain, or a channel.
- SpamAssassin can be configured to use other online databases such as Vipul Razor or Distributed checksum clearinghouse (DCC).
- Messaging Server does not supply an Secure Socket Layer (SSL) version of `libspamass.so`, however, it is possible to build SpamAssassin to use openSSL.
- Perl 5.6 or later is required.

Where Should You Run SpamAssassin?

SpamAssassin can run on a separate system of its own, on the same system as the Messaging Server in a single system deployment, or on the same system as the MTA in a two-tier deployment. If Local Mail Transfer Protocol (LMTP) is used between the MTA and the message store, the filtering must be invoked from the MTA. It cannot be invoked from the message store. When SMTP is used between the MTA and the message store, it can be invoked from either one, and it can run on either system or a separate third system.

If you want to use a farm of servers running SpamAssassin, you would have to use a load balancer in front of them. The MTA is configured with only one address for the SpamAssassin server.

Deploying SpamAssassin

Perform the following steps to deploy SpamAssassin:

- **Install and configure SpamAssassin.** Refer to the SpamAssassin software documentation for installation and configuration information. See also SpamAssassin Options.
- **Load and configure the SpamAssassin client library.** This involves specifying the client library, `libspamass.so`, and configuration file to the MTA (you must create this file). See Loading and Configuring the Spam Filtering Software Client Library.
- **Specify what messages to filter for spam.** Messages can be filtered by user, domain, or channel. See Specifying the Messages to Be Filtered.
- **Specify what actions to take on spam messages.** Spam can be discarded, filed into a folder, tagged on the subject line, and so on. See Specifying Actions to Perform on Spam Messages.
- **Set miscellaneous filter configuration parameters as desired.** See Table.

SpamAssassin Configuration Examples

This section describes some common SpamAssassin configuration examples:

- To Configure Per-user SpamAssassin Scanning
- To File Spam to a Separate Folder
- To Add a Header Containing SpamAssassin Score to Spam Messages
- To Add the SpamAssassin Result String to the Subject Line

> **Note**
> These examples use a number of options and keywords. Refer to Configuring Channel Definitions and Table.

To Configure Per-user SpamAssassin Scanning

SpamAssassin provides the ability to produce a spam score on a per-user basis. The SpamAssassin Network Protocol has the provision to specify an optional *User:* field, which can be set to a constant value by using the `USERNAME` spam filter option.

A new spam filter option `USERNAME_MAPPING` has been added to the SpamAssassin plugin starting with Messaging Server 6.3p1. You use this option to specify the name of a mapping table to probe with address information as the plugin receives recipient addresses from the MTA. The probe format is:

```
current-username|current-recipient-address|current-optin-string
```

Both the `current-optin-string` and the preceding vertical bar are omitted if no `optin` value is specified.

If the mapping sets the `$Y` flag the output string is taken to be the updated username to pass to `spamd`.

> **Note**
> Per-user SpamAssassin scanning can substantially increase the resources required to process emails due to emails addressed to multiple recipients being scanned multiple times.

Per-user SpamAssassin Scanning Example

The following example creates a new channel that is configured to split emails into a single recipient-per-email and then pass the email to SpamAssassin by using the Messaging Server SpamAssassin plugin. The *User:* spam filter option is set to the recipient address. Only recipients who have the `mailConversionTag: peruserspam` attribute have their email sent to the new channel. This enables you to provide per-user scanning to just a subset of users.

The setup and configuration of SpamAssassin to process emails on a per-user basis is not described in this information.

1. Enable the SpamAssassin plugin.
   a. Add the following to *msg_base*`/config/option.dat` file:

```
!
! Spamfiltering settings
spamfilter1_config_file=IMTA_TABLE:spamassassin.opt
spamfilter1_library=IMTA_LIB:libspamass.so
spamfilter1_optional=1
spamfilter1_string_action=data:, require
["editheader","spamtest"]; \
spamadjust "$U"; addheader "X-Spam-Score: $U"
```

b. Add the following to the *msg_base*`/config/spamassassin.opt` file:

```
! Enable debug if set to 1 or 2
DEBUG=0
! This host setting should match the hostname/interface spamd
process is listening on
HOST=127.0.0.1
! This port setting should match what spamd listens on, by
default its 783
PORT=783
! Return a result regardless of whether email is spam or not
MODE=2
! Need to have an empty field, otherwise spamadjust "$U"
doesn't work
FIELD=
! Verdict not used with MODE=2
VERDICT=
! Return rules hit with USE_CHECK=0
!USE_CHECK=0
USERNAME_MAPPING=SPAM_USER
! Default username to use if USERNAME_MAPPING fails to return
a value
USERNAME=default
```

2. Create the required mapping table entry by adding the following to the *msg_base* `/config/mappings` file to use the recipient email address as the spamd *User:* setting:

```
SPAM_USER

! current-username|current-recipient-address|current-optin-string
! no username set
|*|spam $Y$0
! USERNAME=<username> set in spam plugin configuration file
*|*|spam $Y$1
```

3. Create a new channel that splits emails into single recipient per email and sends emails to be scanned by adding the following channel definition to *msg_base*`/config/imta.cnf` file:

```
!
! conversion_peruser
conversion_peruser single sourcespamfilter1optin spam slave_debug
conversion_peruser-daemon
```

4. Create the `CONVERSIONS` mapping table to send emails to set the *new-channel* based on channel tag, by adding the following mapping table entry to the *msg_base*`/config/mappings` file:

```
CONVERSIONS

IN-CHAN=tcp_*;OUT-CHAN=*;TAG=*peruserspam*;CONVERT
Yes,Channel=conversion_peruser
```

5. Define the users that you want to be scanned on a per-user basis by adding the following LDAP attribute to any users you want to have scanning performed on a per-user basis.

```
mailConversionTag: peruserspam
```

Per-user SpamAssassin Scanning Example Results

The following output is from the `spamd` process when sending email to a user with the `mailConversionTag: peruserspam` set. In this output, the user is the <email address>.

```
Jul 17 12:03:17 localhost spamd[5867]: spamd: connection from localhost
[127.0.0.1] at port 35819
Jul 17 12:03:17 localhost spamd[5867]: spamd: checking message
<0JLA00202W56KU00@localhost> for test.user@sun.com:0
Jul 17 12:03:17 localhost spamd[5867]: spamd: clean message (0.6/5.0)
for test.user@sun.com:0 in 0.2 seconds, 773 bytes.
Jul 17 12:03:17 localhost spamd[5867]: spamd: result: . 0 -
AWL,NO_REAL_NAME,UNPARSEABLE_RELAY scantime=0.2,size=773,
user=test.user@sun.com,uid=0,required_score=5.0,rhost=localhost,raddr=127.0
```

The following output shows what to expect in the `mail.log_current` file when per-user scanning is occurring:

```
17-Jul-2007 12:16:13.23 tcp_intranet conversion_peruser EE 1
rfc822;test.user@sun.com
@testserver.sun.com.lmtp:testuser@lmtpcs-daemon
17-Jul-2007 12:16:13.56 conversion_peruser tcp_lmtpcs E 1
rfc822;test.user@sun.com
@testserver.aus.sun.com.lmtp:testuser@lmtpcs-daemon
17-Jul-2007 12:16:13.57 conversion_peruser D 1 rfc822;test.user@sun.com
@testserver.aus.sun.com.lmtp:testuser@lmtpcs-daemon
17-Jul-2007 12:16:13.56 tcp_lmtpcs DL 1 rfc822;test.user@sun.com
@testserver.sun.com.lmtp:testuser@lmtpcs-daemon dns;testserver.sun.com
(testserver.sun.com --
Server LMTP [Sun ONE Messaging Server 6.3-2.01 [built Jun 13 2007;
32bit]]) lmtp;250 2.1.5 testuser@lmtpcs-daemon
and options OK
```

To File Spam to a Separate Folder

This example tests messages arriving at the local message store and files spam into a folder called `spam`. The first three steps can be done in any order.

1. Create the SpamAssassin configuration file.

The name and location of this file is specified in Step 2. A good name is `spamassassin.opt`. This file contains the following lines:

```
host=127.0.0.1
port=2000
mode=0
verdict=spam
debug=1
```

`host` and `port` specify the name of the system where `spamd` is running and the port on which `spamd` listens for incoming requests. `mode=0` specifies that a string, specified by `verdict`, is returned if the message is perceived as spam. `debug=1` turns on debugging in the SpamAssassin library. See Table.

2. Add the following lines to the `option.dat` file:

```
! for Spamassassin
spamfilter1_config_file=/opt/sun/comms/messaging/spamassassin.opt
spamfilter1_library=/opt/sun/comms/messaging/lib/libspamass.so
spamfilter1_optional=1
spamfilter1_string_action=data:,require "fileinto"; fileinto "$U";
```

`spamfilter1_config_file` specifies the SpamAssassin configuration file.
`spamfilter1_library` specifies the SpamAssassin shared library.
`spamfilter1_optional=1` specifies that the MTA continue operation if there is a failure by spamd.
`spamfilter1_string_action` specifies the Sieve action to take for a spam messages.
In this example, `spamfilter1_string_action` is not necessary because the default value already is `data:,require "fileinto"; fileinto "$U";`. This line specifies that spam messages are sent to a folder. The name of the folder is the spam verdict value returned by SpamAssassin. The value returned by SpamAssassin is specified by the `verdict` option in `spamassassin.opt`. (See Step 1.) In this case, the folder name is `spam`.

3. Specify the messages to be filtered.
To filter all messages coming into the local message store, change the `imta.cnf` file by adding the `destinationspamfilter`*X*`optin spam` keywords on the `ims-ms` channel:

```
!
! ims-ms
ims-ms defragment subdirs 20 notices 1 7 14 21 28 backoff "pt5m"
"pt10m" \
"pt30m" "pt1h" "pt2h" "pt4h" maxjobs 2 pool IMS_POOL fileinto \
$U+$S@$D destinationspamfilter1optin spam
ims-ms-daemon
```

4. Recompile the configuration and restart the server. Only the MTA needs to be restarted. You do not need to execute `stop-msg`.

```
# imsimta cnbuild
# imsimta restart
```

5. Start the `spamd` daemon. This is normally done with a command of the form:
`spamd -d`
`spamd` defaults to only accepting connections from the local system. If SpamAssassin and

Messaging Server are running on different systems, this syntax is required:
`spamd -d -i` *listen_ip_address* `-A` *allowed_hosts*
where *listen_ip_address* is the address on which to listen and *allowed_hosts* is a list of authorized hosts or networks (using IP addresses) which can connect to this `spamd` instance.

> **Note**
> 0.0.0.0 can be used with `-i` *listen_ip_address* to have `spamd` listen on all addresses. Listening on all addresses is preferable because it avoids having to change command scripts when changing a system's IP address.

To Add a Header Containing SpamAssassin Score to Spam Messages

This example adds the header `Spam-test:` *result string* to messages determined to be spam by SpamAssassin. An example header might be:

```
Spam-test: True ; 7.3 / 5.0
```

where `Spam-test:` is a literal and everything after that is the result string. `True` means that it is spam (`false` would be not spam). `7.3` is the SpamAssassin score. `5.0` is the threshold. This result is useful for setting up a Sieve filter that can file or discard mail above or between a certain score.

In addition, setting `USE_CHECK` to `0` returns the list of SpamAssassin tests that matched along with the verdict string. See USE_CHECK in Table.

1. Specify the messages to be filtered. This is described in Step 3 in To File Spam to a Separate Folder.
2. Create the SpamAssassin configuration file.
   The name and location of this file is specified with `spamfilter_config`*X*`_file` (see next step). It consists of the following lines:

   ```
   host=127.0.0.1
   port=2000
   mode=1
   field=
   debug=1
   ```

   `host` and `port` specify the name of the system where `spamd` is running and the port on which `spamd` listens for incoming requests. `mode=1` specifies that the SpamAssassin result string is returned if the message is found to be spam. `field=` specifies a string prefix for the SpamAssassin result string. In this example, a prefix is not desired because we are specifying it in the Sieve script. `debug=1` turns on debugging in the SpamAssassin library.

3. Add the following lines to the `option.dat` file:

   ```
   !for Spamassassin
   spamfilter1_config_file=/opt/sun/comms/messaging/config/spamassassin.
   ["editheader"];addheader "Spam-test" "$U";
   ```

   As in previous examples, the first three options specify the SpamAssassin configuration file, shared library, and to continue MTA operation if there is a failure by the shared library. The following line:

   ```
   spamfilter1_string_action=data:,require ["editheader"];addheader
   "Spam-test" "$U";
   ```

specifies that a header is added to spam messages. The header has the literal prefix Spam-text: followed by the string returned by SpamAssassin. Because mode=1 was specified in the previous step, the SpamAssassin result string is returned. For example: True; 7.3/5.0

4. Recompile the configuration, restart the server and start the `spamd` daemon.
   See SpamAssassin Configuration Examples.

To Add the SpamAssassin Result String to the Subject Line

By adding the SpamAssassin result string to the Subject line, users can determine whether they wish to read a message with a SpamAssassin score. For example:

```
Subject: [SPAM True ; 99.3 / 5.0] Free Money At Home with Prescription
Xanirex!
```

Note that setting `USE_CHECK` to `0` returns the list of SpamAssassin tests that matched along with the verdict string (see SpamAssassin Options). This list can be very long, so it is best to set `USE_CHECK` to `1`.

1. Specify the messages to be filtered.
   See Step 3 in To File Spam to a Separate Folder.
2. Create the SpamAssassin configuration file.
   This step is described in To File Spam to a Separate Folder. `mode=1` specifies that the SpamAssassin result string is returned if the message is found to be spam.

   ```
   host=127.0.0.1
   port=2000
   mode=1
   debug=1
   ```

   `host` and `port` specify the name of the system where `spamd` is running and the port on which `spamd` listens for incoming requests. `mode=1` specifies that the SpamAssassin result string is returned if the message is spam. debug=1 turns on debugging in the SpamAssassin library.

3. Add the following lines to the `option.dat` file:

   ```
   !for Spamassassin
   spamfilter1_config_file=/opt/sun/comms/messaging/config/spamassassin.
   "[SPAM detected: $U]";
   ```

   As in previous examples, the first three options specify the SpamAssassin configuration file, shared library, and to continue MTA operation if there is a failure by the shared library. The following line
   `spamfilter1_string_action=data:,addtag "[SPAM detected $U]"`
   specifies that a tag be added to the `Subject:` line. It has the literal prefix SPAM `detected` followed by the field string (default: `Spam-Test`) followed by "[*result string*]" returned by SpamAssassin. Because `mode=1` was specified in SpamAssassin Configuration Examples, the SpamAssassin result string is returned. Thus, a subject line looks something like this:
   `Subject: [SPAM detected Spam-Test: True ; 11.3 / 5.0] Make Money!`
   You can also use `addheader` and `addtag` together:
   `spamfilter1_string_action=data:,require ["editheader"];addtag "[SPAM detected $U]";addheader "Spamscore" "$U";`
   to get a message like this:
   `Subject: [SPAM detected Spam-Test: True ; 12.3 / 5.0] Vigaro Now!`
   `Spamscore: Spam-Test: True ; 12.3 / 5.0`

Set `field=` in `spamassassin.opt` to remove the default value of `Spam-Test`. A cleaner message is returned:

```
Subject: [SPAM True ; 91.3 / 5.0] Vigaro Now!
Spamscore: True ; 91.3 / 5.0
```

4. Recompile the configuration, restart the server and start the `spamd` daemon.
   See To File Spam to a Separate Folder.

To Filter Messages Based on SpamAssassin Score

This example shows how to filter messages based on a SpamAssassin score. It uses the `spamadjust` and `spamtest` Sieve filter actions. In this example, a header containing the SpamAssassin score is added to all messages. This header can be used by the SpamAssassin software administrator to tune SpamAssassin for improved spam email detection. If the message has a SpamAssassin score between 5 and 10, the message is filtered to a `spam` folder within the user's account. If the message has a SpamAssassin score greater then 10, the message is discarded. Note that by default SpamAssassin considers messages with a score of 5 and greater to be spam.

1. Specify the messages to be filtered.
   This is described in Step 3 of To File Spam to a Separate Folder.
2. Create the SpamAssassin configuration file.
   The name and location of this file is specified with `spamfilter_config`*X*`_file` (see next step). It consists of the following lines:

   ```
   debug=1
   host=127.0.0.1
   port=783
   mode=2
   field=
   ```

   `host` and `port` specify the name of the system where `spamd` is running and the port on which `spamd` listens for incoming requests. `mode=2` specifies that the SpamAssassin result string is always returned regardless of the score. `field=` specifies a string prefix for the SpamAssassin result string. In this example, a prefix is not desired because we are specifying it in the Sieve script. `debug=1` turns on debugging in the SpamAssassin library.

3. Add the following lines to the option.dat file

   ```
   ! For SpamAssassin
   spamfilter1_config_file=/opt/sun/comms/messaging/config/spamassassin.
   require ["editheader","spamtest"]; \
   spamadjust "$U"; addheader "Spam-test" "$U";
   ```

   As in previous examples, the first three lines specify the SpamAssassin configuration file, shared library, and to continue MTA operation if there is a failure by the shared library. The last two lines specify that the SpamAssassin score should be extracted from the return string from SpamAssassin ($U), which is used in the `spamtest` operation, and a spam score header should be added to all messages (for example, `Spam-test: True; 7.3/5.0`)

4. Create a channel level filter to process the email based on the spam score.
   Refer to To Create a Channel-level Filter. Add the following rule to that file:

```
require
["spamtest","relational","comparator-i;ascii-numeric","fileinto"];
if spamtest :value "ge" :comparator "i;ascii-numeric" "10"
{discard;}
elsif spamtest :value "ge" :comparator "i;ascii-numeric" "5"
{fileinto "spam";}
else {keep;}
```

The second line discards the spam email if the SpamAssassin score is greater or equal to 10. The third line files the email to the users "spam" folder if the score is greater or equal to 5. The last line `else {keep;}` keeps all messages which received a score less then 5.

5. Recompile the configuration, restart the server and start the `spamd` daemon
   See the final steps in To File Spam to a Separate Folder.

Testing SpamAssassin

To test SpamAssassin, first set `debug=1` in the `spamassassion.opt` file. You do not have to turn on the channel-specific `master_debug` or `slave_debug` in the `imta.cnf`. Then send a test message to a test user. The *msg-svr-base*`/data/log/tcp_local_slave.log*` file should have lines similar to these:

```
15:15:45.44: SpamAssassin callout debugging enabled; config
/opt/sun/comms/messaging/config/spamassassin.opt
15:15:45.44: IP address 127.0.0.1 specified
15:15:45.44: Port 2000 selected
15:15:45.44: Mode 0 selected
15:15:45.44: Field "Spam-Test: " selected
15:15:45.44: Verdict "spam" selected
15:15:45.44: Using CHECK rather than SYMBOLS
15:15:45.44: Initializing SpamAssassin message context
...
15:15:51.42: Creating socket to connect to SpamAssassin
15:15:51.42: Binding SpamAssassin socket
15:15:51.42: Connecting to SpamAssassin
15:15:51.42: Sending SpamAssassin announcement
15:15:51.42: Sending SpamAssassin the message
15:15:51.42: Performing SpamAssassin half close
15:15:51.42: Reading SpamAssassin status
15:15:51.67: Status line: SPAMD/1.1 0 EX_OK
15:15:51.67: Reading SpamAssassin result
15:15:51.67: Result line: Spam: False ; 1.3 / 5.0
15:15:51.67: Verdict line: Spam-Test: False ; 1.3 / 5.0
15:15:51.67: Closing connection to SpamAssassin
15:15:51.73: Freeing SpamAssassin message context
```

If your log file does not contain lines similar to these, or if `spamd` is not running, the following error message is returned in your SMTP dialog after the last period (.) is sent to the SMTP server.

```
452 4.4.5 Error writing message temporaries - Temporary scan failure: End
message status = -1
```

In addition, if `spamfilter1_optional=1` (highly recommended) is set in `option.dat`, the message

is accepted, but not filtered. It is as if spam filtering was not enabled, and the following lines appear in `tcp_local_slave.log*`:

```
15:35:15.69: Creating socket to connect to SpamAssassin
15:35:15.69: Binding SpamAssassin socket
15:35:15.69: Connecting to SpamAssassin
15:35:15.69: Error connecting socket: Connection refused
15:35:15.72: Freeing SpamAssassin message context
```

The call to SpamAssassin happens after the SMTP server received the entire message (that is, after the last "." is sent to the SMTP server), but before the SMTP server acknowledges to the sender that it accepted the message.

Another test is to send a sample spam message using `sample-spam.txt` from, for example, the `Mail-SpamAssassin-2.60` directory. This message has the following special text string in it:

```
XJS*C4JDBQADN1.NSBN3*2IDNEN*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL*C.34X
```

The corresponding `tcp_local_slave.log*` contains something like this:

```
16:00:08.15: Creating socket to connect to SpamAssassin
16:00:08.15: Binding SpamAssassin socket
16:00:08.15: Connecting to SpamAssassin
16:00:08.15: Sending SpamAssassin announcement
16:00:08.15: Sending SpamAssassin the message
16:00:08.15: Performing SpamAssassin half close
16:00:08.15: Reading SpamAssassin status
16:00:08.43: Status line: SPAMD/1.1 0 EX_OK
16:00:08.43: Reading SpamAssassin result
16:00:08.43: Result line: Spam: True ; 1002.9 / 5.0
16:00:08.43: Verdict line: Spam-Test: True ; 1002.9 / 5.0
16:00:08.43: Closing connection to SpamAssassin
16:00:08.43: Mode 0 verdict of spam
16:00:08.43: Mode 0 verdict of spam
16:00:08.47: Freeing SpamAssassin message context
```

A corresponding entry in the mail.log_current file would look as follows. Note the +spam part of the destination address, which means the message is filed in the folder called spam:

```
15-Dec-2003 15:32:17.44 tcp_intranet ims-ms E 1 morchia@siroe.com
rfc822;
morchia morchia+spam@ims-ms-daemon 15-Dec-2003 15:32:18.53
ims-ms D 1 morchia@siroe.com rfc822;morchia morchia+spam@ims-ms-daemon
```

SpamAssassin Options

This section contains the SpamAssassin option table.

SpamAssassin Options (spamassassin.opt)

Options	Description	Default

`debug`	Specifies whether to turn on debugging in the `libspamass.so`. Debugging of `spamd` itself is controlled by the command line invoking `spamd`. Set to an integer value. 0 is off, 1 is on, a setting of 2 or greater reports exactly what was received from spamd.	0

`field`	Specifies the string prefix for the SpamAssassin result. SpamAssassin results look like this:
	
	`Spam-test: False ; 0.0 / 5.0`
	`Spam-test: True ; 27.7 / 5.0`
	
	The `field` option provides the means for changing the `Spam-test:` part of the result. Note that the ":" is removed if an empty `field` value is specified. If *USE_CHECK* is set to 0, the result string will look similar to this:
	
	`Spam-test: False ; 0.3 / 4.5 ; HTML_MESSAGE,NO_REAL_NAME`
	`Spam-test: True ; 8.8 / 4.5 ; NIGERIAN_BODY, NO_REAL_NAME,PLING_PLING,RCVD_IN_SBL, SUBJ_ALL_CAPS`
		"Spam-test"

`host`	The name of the system where `spamd` is running.	localhost

mode	Controls the translation of SpamAssassin filter results to verdict information. That is, it specifies what verdict information is returned after a message is processed. Four modes are available. See The SpamAssassin mode Option for further explanation. 0 - Return a *verdict string* (specified by the verdict option), if the message is spam. The MTA option spamfilter*X*_string_action can be used to specify what to do if a verdict string is returned. If the verdict option (defined below) is empty or unspecified, and message is spam, a *null verdict* is returned. The MTA option spamfilter*X*_null_action can be used to specify what to do if a null verdict is returned. Returns a *SpamAssassin default result string* if it is not spam. (A default verdict always means to take no action and deliver as normal.) 1 - Returns the SpamAssassin *result string* if the message is found to be spam. Returns a *SpamAssassin default result string* if it is not spam. (Again, a default verdict always means to take no action and deliver as normal.) A SpamAssassin result string looks something like this: `True; 6.5 / 7.3` 2 - Same as mode 1 except that the SpamAssassin result string is returned regardless of whether the message is spam or not spam. No default or null verdict is ever returned and the verdict option is never used. 3 - Return the SpamAssassin result string if the message is found to be spam; return the verdict string specified by the verdict option if it is not. You can control the action for the SpamAssassin result string by using the spamfilter*X*_verdict_*n* and spamfilter*X*_action_*n* matched pair. You can control the action for the verdict string by using spamfilter*X*_string_action.	0
port	Specifies the port number where spamd listens for incoming requests.	783
USE_CHECK	1 - The spamd CHECK command is used to return the SpamAssassin score. 0 - Enables use of the SYMBOLS command which returns a score and a list of the SpamAssassin tests that matched. The system may hang or have other problems with this option in pre-2.55 versions of SpamAssassin. See *field* above.	
SOCKS_HOST	String. Specifies the name of an intermediate SOCKS server. If this option is specified the ICAP connection is made through the specified SOCKS server and not directly.	""
SOCKS_PORT	Specifies the port that the intermediate SOCKS server is running on.	1080

USERNAME_MAPPING	Specify the name of a mapping to probe with address information as the plugin receives recipient addresses from the MTA. The probe format is: `current-username|current-recipient-address|current-optin-string` Both the *current-optin-string* and the preceeding vertical bar are omitted if no optin value was specified. If the mapping sets the $Y flag the output string is taken to be the updated username to pass to spamd.	""
verdict	Specifies the verdict string used for MODE 0.	""

The SpamAssassin mode Option

After processing a message, SpamAssassin determines whether a message is spam or not. mode allows you to specify the string that is returned to indicate the verdict. The string choices are null, default, SpamAssassin result string, or a verdict string specified with the verdict option. (Note that *default* is neither null, the SpamAssassin result string, nor the string specified by verdict, but some other non-configurable result string.) The mode operations are outlined in the table below.

Table 14-4 Returned String for the SpamAssassin mode Option

verdict Setting	Spam?	*mode=0*	*mode=1*	*mode=2*	*mode=3*
verdict="" (not set)	yes	null	SpamAssassin result	SpamAssassin result	SpamAssassin result
	no	default	default	SpamAssassin result	default
verdict=*string*	yes	verdict *string*	SpamAssassin result	SpamAssassin result	SpamAssassin result
	no	default	default	SpamAssassin result	verdict *string*

The first column indicates whether the verdict option is set or not set. The second column indicates whether the message is spam or not. The mode columns indicate the string returned for the various modes. For example, if verdict is not set and mode is set to 0 and a message is not spam, a default string is returned. If the verdict is set to YO SPAM! and mode is set to 0 and a message is spam, the string YO SPAM! is returned.

Using Symantec Anti-Virus Scanning Engine (SAVSE)

In addition to describing how to deploy SAVSE this section can also be useful in deploying other ICAP-supported anti-spam/anti-virus programs. This section consists of the following subsections:

- SAVSE Overview
- SAVSE Requirements and Usage Considerations
- Deploying SAVSE
- SAVSE Configuration Example
- SAVSE Options

SAVSE Overview

SAVSE is a TCP/IP server application and communication Application Programming Interface (API) that

provides virus scanning services. Designed specifically to protect traffic served through, or stored on, network infrastructure devices, it detects and protects against viruses, worms, and Trojan horses in all major file types, including mobile code and compressed file formats. Refer to the Symantec website for detailed information

> **Note**
> The current version of Messaging Server only supports the SAVSE scan function. It does not support the repair or delete functions.

SAVSE Requirements and Usage Considerations

This is a separately licensed product from Symantec.

Only the scan mode is supported, not the scan and repair or scan and delete mode in the SAVSE configuration.

Where Should You Run SAVSE?

SAVSE or another server that supports ICAP can run on a separate system of its own, on the same system as the Messaging Server in a single system deployment, or in a two-tier deployment on the same system as the MTA. If Local Mail Transfer Protocol (LMTP) is used between the MTA and the message store, the filtering must be invoked from the MTA. It cannot be invoked from the message store. When SMTP is used between the MTA and the message store, it can be invoked from either one, and it can run on either system or a separate third system.

If you want to use a farm of servers running SAVSE, you would have to use a load balancer in front of them. The MTA is configured with only one address for the SAVSE server.

Deploying SAVSE

Perform the following steps to deploy SAVSE.

- **Install and configure the SAVSE.** Refer to the Symantec software documentation for installation and configuration information. See also SAVSE Options.
- **Load and configure the SAVSE client library.** This involves specifying the client library `libicap.so` and configuration file to the MTA (you must to create this file). See Loading and Configuring the Spam Filtering Software Client Library.
- **Specify what messages to filter for viruses.** Messages can be filtered by user, domain, or channel. See Specifying the Messages to Be Filtered.
- **Specify what actions to take on virus messages.** Viruses can be discarded, filed into a folder, tagged on the subject line, and so on. See Specifying Actions to Perform on Spam Messages.
- **Set miscellaneous filter configuration parameters as desired.** See Table and Specifying Actions to Perform on Spam Messages.

SAVSE Configuration Example

The following example tests messages arriving at the local message store and discards messages with attached viruses. The first three steps can be done in any order.

To Configure SAVSE

1. Create the SAVSE configuration file.
   The name and location of this file is specified in the next step. The name used here is `SAVSE.opt`. An example of this file is shown below:

```
host=127.0.0.1
port=1344
mode=0
verdict=virus
debug=1
```

`host` and `port` specify the name of the system where the SAVSE program is running and the port (1344 is the default for SAVSE) on which it listens for incoming requests. `mode=0` specifies that a string, specified by `verdict` (in this case the word `virus`), will be returned if the message is perceived to contain a virus. `debug=1` turns on debugging. See SAVSE Options for a description of the ICAP configuration parameters.

2. Create an `option.dat` file. Example:

```
! for Symantex Anti-virus Scan Engine
spamfilter1_config_file=/opt/sun/comms/messaging/config/SAVSE.opt
spamfilter1_library=/opt/sun/comms/messaging/lib/libicap.so
spamfilter1_optional=1
spamfilter1_string_action=data:,discard
```

`spamfilter1_config_files` specifies the SAVSE configuration file.
`spamfilter1_library` specifies the location of the SAVSE shared library.
`spamfilter1_optional=1` specifies that the MTA continue operation if there is a failure by the SAVSE program.
`spamfilter1_string_action` specifies the Sieve action to take for a spam messages. This value specifies that messages with viruses are discarded. Since this is the default value, you don't have to specify it unless you are changing the value.

3. Specify the messages to be filtered.
To filter all messages coming into the local message store, change the `imta.cnf` file by adding the `destinationspamfilter1optin spam` keywords on the `ims-ms` channel:

```
!
! ims-ms
ims-ms defragment subdirs 20 notices 1 7 14 21 28 backoff "pt5m"
"pt10m" \
"pt30m" "pt1h" "pt2h" "pt4h" maxjobs 2 pool IMS_POOL fileinto \
$U+$S@$D destinationspamfilter1optin virus
ims-ms-daemon
```

4. Recompile the configuration and restart the server. Only the MTA needs to be restarted. You do not need to execute `stop-msg`.

```
# imsimta cnbuild
# imsimta restart
```

5. Make sure SAVSE is started.
It should have started automatically, but if not, the start command might looks something like this:
`/etc/init.d/symcscna start`

Other Possible Configurations

Setting `mode` to `0` can be used with a `spamfilter`*X*`_null_option` to take some other action, such as

filing messages in a particular folder when they are determined to be spam. For example:

```
spamfilter1_null_option=data:,require "fileinto"; fileinto "VIRUS";
```

Note that filing infected messages into a folder is not a good idea in most cases.

Setting `mode` to 1 can be used to start an action. For example, the spam result could be included in the reject message by setting `mode` to 1 and the `spamfilter`*X*`_string_action` option in the MTA to something like:

```
spamfilter1_string_action=data:,require "reject"; reject "Message contained a
virus [$U]";
```

Like `fileinto`, using the `reject` action to deal with viruses is rarely a good idea because it sends the virus back to the sender.

You could also add a tag to the spam message header by adding a line to the `option.dat` file. Example:

```
spamfilter1_string_action=data:,addtag "[SPAM detected!]";
```

Setting `mode` to 2 can be used where an action needs to be taken regardless of whether or not the message was determined to contain a virus. The addition of a header field that can subsequently be tested is an obvious application for mode 2:

```
spamfilterX_string_action=data:,require ["editheader"];addheader "$U"
```

SAVSE Options

The SAVSE option file is really a more generic ICAP option file. Its name and location is set by `spamfilter`*X*`_config_file` in `option.dat`. It consists of lines of the form `option=value`. The one required option is `HOST`. It must be set to the name of system where the ICAP filtering server is running. This option must be set even if the ICAP server is running on the local host. The option file is shown below.

ICAP Options

Options	Description	Default
`debug`	Enables or disables debug output from the ICAP interface module. 0 or 1.	0

`field`	Specifies the prefix for the ICAP result. SAVSE result strings look like this:
	
	`Virus-Test: False`
	`Virus-Test: True; W32.Mydoom.A@mm.enc`
	
	This option provides a way to change the `Virus-Test:` part of the result. Note that the ": " is removed if an empty `field` value is specified.
		Virus-Test

`host`	The name of the system where the ICAP filtering server is running	localhost

mode	Controls the translation of ICAP filter results to verdict information. That is, it specifies the string information returned after a message is processed. Four modes are available. See The ICAP mode Option for further explanation 0 - Returns a *verdict string* (specified by the verdict option), if the message contains a virus. The MTA option spamfilter*X*_string_action can be used to specify what to do if a verdict string is returned. If the verdict option is empty or not set, a *null verdict* is returned. The MTA option spamfilter*X*_null_action can be used to specify what to do if a null verdict is returned and if you want to override the default action, which is to discard the message. If the message does not contain a virus, a default string is returned. A default string is unconfigurable and always means to take no action and deliver as normal. 1 - Return the ICAP *result string* if the message is found to contain a virus. If the message does not contain a virus, a default string is returned. A default string always means to take no action and deliver as normal. Below are two examples of a ICAP result string: VIRUS TEST: FALSE VIRUS-TEST: TRUE; W32.Mydoom.A@mm.enc 2 - Return an ICAP result string unconditionally; no default or null verdict is ever returned and the verdict option is never used. This setting is intended for cases in which an action needs to be taken regardless of whether or not the message was determined to contain a virus. The addition of a header field that can subsequently be tested is an obvious application for mode 2: spamfilterX_string_action=data:,require ["editheader"]; addheader "$U" 3 - Return the ICAP result string if the message is found to contain a virus; return the verdict string specified by the verdict option if it does not. This setting is intended for cases in which one action needs to be taken if a virus is found and another taken if one is not. You can control the action for the ICAP result string by using the spamfilter*X*_verdict_*n* and spamfilter*X*_action_*n* matched pair. You can control the action for the verdict string by using spamfilter*X*_string_action.	0
port	Specifies the port number on which the ICAP server is running.	1344
SOCKS_HOST	String. Specifies the name of an intermediate SOCKS server. If this option is specified, the ICAP connection is made through the specified SOCKS server and not directly.	""
SOCKS_PORT	Integer. Specifies the port on which the intermediate SOCKS server is running.	1080
verdict	Specifies the verdict string used for MODE 0 and 3.	""

The ICAP mode Option

After processing a message, ICAP anti-virus programs like SASVE determines whether a message has a virus or not. mode allows you to specify the string returned by the ICAP program indicating this verdict. The string choices are *null*, *default*, *ICAP result string*, or a verdict *string* (specified with the verdict option). Note that *default* is not null, the ICAP result string, nor the string specified by verdict, but some other non-configurable string returned by the program. The mode operations are outlined in the following table.

Returned Verdict String for the ICAP mode Option

verdict\Setting	Virus?	*mode=0*	*mode=1*	*mode=2*	*mode=3*
verdict="" (not set)	yes	null	ICAP result	ICAP result	ICAP result
	no	default	default	ICAP result	default
verdict=string	yes	verdict *string*	ICAP result	ICAP result	ICAP result
	no	default	default	ICAP result	verdict *string*

The first column indicates whether the `verdict` option is set or not set. The second column indicates whether the message contains a virus or not. The mode columns indicate the string returned for the various modes. For example, if `verdict` is not set and `mode` is set to 0 and a message does not have a virus, the ICAP program returns a default. If the `verdict` is set to `WARNING VIRUS!` and `mode` is set to 0 and a message does have a virus, the ICAP program returns the string `WARNING VIRUS!`.

Using ClamAV

Messaging server supports the use of the popular and freely available third-party virus scanner ClamAV for the detection of virus- and Trojan horse- infected messages. Virus signatures used by ClamAV to detect newly created viruses can be automatically updated using the `freshclam` utility provided with the ClamAV software package.

Further information on ClamAV can be found at the ClamAV website.

ClamAV/Messaging Server Theory of Operations

ClamAV integration in Messaging Server makes use of the `clamd` daemon that is provided as part of the ClamAV package. `clamd` is a multi-threaded process that listens on a socket for requests to process messages. After processing the message, it sends back a response and closes the connection. The client portion, `clamdscan` from the ClamAV installation, is not used. This function is done by a shared library called `libclamav.so`, which is part of Messaging Server.

`libclamav.so` is loaded the same way as the Brightmail SDK is loaded.

ClamAV Requirements and Usage Considerations

ClamAV can run on a separate system of its own, on the same system as the Messaging Server in a single system deployment, or on the same system as the MTA in a two-tier deployment. If Local Mail Transfer Protocol (LMTP) is used between the MTA and the message store, the filtering must be invoked from the MTA. It cannot be invoked from the message store. When SMTP is used between the MTA and the message store, it can be invoked from either one.

If you want to use a farm of servers running ClamAV, use a load balancer front of them. The MTA is configured with only one address for the ClamAV server.

Other considerations.

- ClamAV is free. Go to http://clamav.net for software and documentation.
- ClamAV integration with the MTA can be enabled for a user, a domain, or a channel.
- The ClamAV package provides a utility to regularly update virus-signatures. The utility is called `freshclam`. Refer to the ClamAV package documentation for further information.
- The `libclamav.so` library is included by default with Messaging Server 6.2 and above.

Deploying ClamAV

Perform the following steps to deploy ClamAV:

- **Install and configure ClamAV.** Refer to the ClamAV software documentation for installation and configuration information. See also ClamAV Options.
- **Load and configure the ClamAV client library.** This involves specifying the client library, `libclamav.so` and a configuration file to the MTA (you must create this file). See Loading and Configuring the Spam Filtering Software Client Library.
- **Specify what messages to filter for spam.** Messages can be filtered by user, domain, or channel. See Specifying the Messages to Be Filtered.
- **Specify what actions to take on virus messages.** See Specifying Actions to Perform on Spam Messages.
- **Set miscellaneous filter configuration parameters as desired.** See ClamAV Options.

To Jettison Virus - or Trojan Horse - Infected Email Using ClamAV

The following example jettisons all messages found to contain a virus or Trojan horse detected by ClamAV. The verdict string is not used.

1. Create the ClamAV configuration file.
   The name and location of this file is specified in Step 2. A good name is `clamav.opt`. This file contains the following lines:

   ```
   ! ClamAV Settings
   debug=1
   host=127.0.0.1
   port=3310
   mode=1
   ```

   `debug=1` turns on debugging in the ClamAV library.
   `host` and `port` specify the name of the system where `clamd` is running and the port on which `clamd` listens for incoming requests.
   `mode=1` specifies that the ClamAV plug-in return the ClamAV result string as the verdict when a virus infected email is detected.

2. Modify the `option.dat` file.
   Add the following lines to the `option.dat` file:

   ```
   ! ClamAV settings
   spamfilter2_config_file=/opt/sun/comms/messaging/config/clamav.opt
   spamfilter2_library=/opt/sun/comms/messaging/lib/libclamav.so
   spamfilter2_string_action=data:,require ["jettison"]; jettison;
   ```

   `spamfilter2_config_file` specifies the ClamAV configuration file.
   `spamfilter2_library` specifies the ClamAV shared library.
   `spamfilter2_string_action` specifies the Sieve action to take for a virus infected email.

3. Specify the messages to be filtered.
   To filter all messages coming into the local message store, change the `imta.cnf` file by adding the `destinationspamfilterXoptin` virus keywords on the `ims-ms` channel:

```
!
! ims-ms
ims-ms defragment subdirs 20 notices 1 7 14 21 28 backoff "pt5m"
"pt10m" \
"pt30m" "pt1h" "pt2h" "pt4h" maxjobs 2 pool IMS_POOL fileinto \
$U+$S@$D destinationspamfilter2optin virus
ims-ms-daemon
```

4. Recompile the configuration and restart the server.
   Only the MTA needs to be restarted. You do not need to execute stop-msg.

```
# imsimta cnbuild
# imsimta restart
```

5. Start the `clamd` daemon.

Testing ClamAV

To test ClamAV, first set `debug=1` in the `clamav.opt` file. (You do not have to turn on the channel-specific `master_debug` or `slave_debug` in the `imta.cnf`.) Then send a file attachment to a test user which contains the EICAR virus string (http://www.eicar.org/anti_virus_test_file.htm). This string is designed to trigger virus scanners to recognize an email as virus-infected without having an actual virus attached:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Review the test logs. The *msg-svr-base*`/data/log/tcp_local_slave.log*` file should have lines similar to these:

```
10:39:00.85: ClamAV callout debugging enabled;
config /opt/sun/comms/messaging/config/clamav.opt
10:39:00.85: IP address 127.0.0.1 specified
10:39:00.85: Port 3310 selected
10:39:00.85: Mode 1 selected
10:39:00.85: Field "Virus-Test: " selected
10:39:00.85: Verdict "" selected
10:39:00.85: Initializing ClamAV message context
...
10:39:00.85: Creating socket to connect to clamd server
10:39:00.85: Binding clamd socket
10:39:00.85: Connecting to clamd server
10:39:00.85: Sending ClamAV STREAM request
10:39:00.85: Retrieving ClamAV STREAM response
10:39:00.85: STREAM response: PORT 2003
10:39:00.85: Creating socket to connect to clamd server data port
10:39:00.85: Binding clamd data socket
10:39:00.85: Connecting to clamd server data port
10:39:00.85: Sending ClamAV the message
10:39:00.85: Closing ClamAV data connection
10:39:00.85: Reading ClamAV result
10:39:00.87: Result line: stream: Eicar-Test-Signature FOUND
10:39:00.87: Scan result: Message is infected
10:39:00.87: Verdict line: Virus-Test: True ; Eicar-Test-Signature
10:39:00.87: Closing connection to ClamAV
10:39:00.87: Mode 1 verdict of Virus-Test: True ; Eicar-Test-Signature
10:39:00.87: Mode 1 verdict of Virus-Test: True ; Eicar-Test-Signature
...
10:39:00.87: Freeing ClamAV message context
```

If your log file does not contain lines similar to these, or if `clamd` is not running, the following error message is returned in your SMTP dialog after the last period (.) is sent to the SMTP server:

```
452 4.4.5 Error writing message temporaries - Error connecting to ClamAV
server
```

ClamAV Options

The ClamAV option file is a typical messaging server-style option file consisting of lines of the form `option=value`. The one required option is `HOST`. It must be set to the name of the system where `clamd` is running. This option must be set even if `clamd` is running on the local host.

Further additional options are available for this options file are shown in the following table.

ClamAV Options

Option	Description	Default

DEBUG	Enables or disables debug output from the ClamAV interface module. (Debug output from `clamd` itself is controlled by options on the `clamd` command line.) The larger the value, the more debugging output will be produced. 0 produces no output. 1 provides basic debugging. 2 adds logging of TCP traffic from `clamd`.	0

FIELD	Specifies the ClamAV result string prefix. ClamAV result strings generally look something like one of the following:
	
	`Virus-Test: False`
	`Virus-Test: True ; Worm.Mydoom.I`
	
	The `FIELD` option provides the means for changing the `Virus-Test` part of the result. Note that the ": " will also be removed if an empty `FIELD` value is specified.
		"Virus-Test"

USE_INSTREAM	(New in MS7u2) Enables (1) or disables (0) the use of INSTREAM scanning in clamd. Starting with ClamAV 0.95, clamd fixed the deficient design that required two TCP connections to be made when scanning a data stream (control connection + data connection). Now only a single connection is needed which allows for simpler and faster processing (no setup/processing/teardown of data connection) and also allows ClamAV to scale horizontally by placing multiple scanning systems behind load balancers. Attempting to use this option with an earlier version of ClamAV will result in a scanning failure.	0
MESSAGE_BUFFER_SIZE	Due to the nature of the `clamdscan/clamd` interface the ClamAV plugin has to buffer the message in memory before sending to ClamAV. The size of the memory buffer is controlled by this option. It defaults to 1,048,576 characters. Messages longer than this will be truncated and not sent in their entirety to ClamAV. In order to ensure that every message is scanned fully, this value should reflect the maximum message size the MTA will accept. Reducing this value may help to speed up virus scanning times, but may let through viruses undetected.	1048576

MODE	Controls the translation of ClamAV results to verdict information. Four different modes are available: 0 - Return the verdict string specified by the VERDICT option if the message is found to contain a virus; return a default verdict if it does not. A null verdict is returned if the VERDICT option is empty or unspecified. 1 - Return the ClamAV result as a verdict if the message is found to contain a virus; return a default verdict if not. 2 - Return a ClamAV result string as the verdict unconditionally; no default or null verdict is ever returned and the VERDICT option is never used. 3 - Return the ClamAV result as a verdict if the message is found to contain a virus; return the verdict string specified by the VERDICT option if it is not.	0
PORT	Specifies the port clamd is running on.	3310
SOCKS_HOST	Specifies the name of an intermediate SOCKS server. If this option is specified the clamd connection is made through the specified SOCKS server and not directly.	3310
SOCKS_PORT	Specifies the port the intermediate SOCKS server is running on.	1080
VERDICT	Specifies the verdict string used in modes 0 and 3.	""

Support for Sieve Extensions

In addition to the standard Sieve functions, Messaging Server provides support for a number extensions including addheader, addtag, spamtest and spamadjust. addheader and addtag are described in To Add a Header Containing SpamAssassin Score to Spam Messages and To Add the SpamAssassin Result String to the Subject Line.

These extensions gives administrators the ability to set different threshold values as well as the ability to set up white lists that override SpamAssassin verdicts. The two can even be combined to have different thresholds depending on who sent a particular message. spamadjust is a non-standard action. spamtest is described in http://www.ietf.org/rfc/rfc3685.txt. Note that both can be used with Symantec Brightmail as well as SpamAssassin.

The internal separator character used to delimit multiple subject line tag additions for addtag has been changed from space to vertical bar. This makes it possible to add a tag containing spaces, as some spam filters want to do. For example, previously, addtag "[Probable Spam]" was taken to mean addtag "[Probable" and addtag "spam]". Now it is seen as a single tag, "[Probable Spam]". This change prevents vertical bars from being used in tags.

spamtest can be used to compare SpamAssassin scores to specific values using the Sieve [RELATIONAL] extension with the "i;ascii-numeric" comparator. The SpamAssassin score is typically a real number, but spamtest forces the score into an integer value between 0 an 10 by first rounding the score to the nearest integer. Values below 0 are forced up to 0 while values above 10 are forced down to 10. Finally, the text string maintained by Messaging Server is appended to produce the test string that the spamtest test sees. :percent is supported by spamtest (see SIEVE Email Filtering: Spamtest and Virustest Extensions draft-ietf-sieve-spamtestbis-05). Note that {{spamtest}}can be also be used with Brightmail.

`spamadjust` is used to adjust the current spam score. This action takes a single string argument which is scanned for a real numeric value. This value is used to adjust the current spam score. The entire string is also appended to the current score text string. In the example shown below, the string would be `"undisclosed recipients"`.

Multiple `spamadjust` actions are allowed; each one is added to the current score. Again, the score value always starts at 0. Signed numeric values are allowed, making it possible to lower the current score as well as raise it. There is no `require` clause for `spamadjust`; the `spamtest` extension should be listed instead.

For example, a possible use of `spamadjust` with a SpamAssassin `MODE` setting of 2 might be:

`spamfilter`*X*`_string_action=data:,require ["spamtest"];spamadjust "$U";`

A system-level Sieve filter could then modify the SpamAssassin score by checking for a particular type of header and, if found, adding 5 to the SpamAssassin score:

```
require "spamtest";
if header :contains ["to", "cc", "bcc", "resent-to", "resent-cc",
"resent-bcc"] ["<undisclosed recipients>", "undisclosed.recipients"]
{spamadjust "+5 undisclosed recipients";}
```

Note that `spamadjust` can be also be used with Brightmail.

Finally, a user-level Sieve script could test the resulting value, discard messages certain to be spam, file messages likely to be spam, and allow messages from addresses in the local domain to pass through:

```
require ["spamtest", "relational",
"comparator-i;ascii-numeric", "fileinto"];
if anyof (address :matches "from" ["*@siroe.com", "*@*.siroe.com"])
{keep;}
elsif spamtest :value "ge" :comparator "i;ascii-numeric" "8"
{discard;}
elsif spamtest :value "ge" :comparator "i;ascii-numeric" "5"
{fileinto "spam-likely";}
else
{keep;}
```

Using Milter

This section consists of the following subsections:

- Milter Overview
- Milter/Messaging Server Theory of Operations
- Milter Requirements and Usage Considerations
- Per-recipient Modification Actions
- To Deploy Milter

Milter Overview

Milter is the short name for Sendmail Content Management API. It also refers to software written using this API. Milter provides a plug-in interface for third-party software to validate, modify, or block messages as they pass through the MTA. Milters can process a message's connection (IP) information, envelope

protocol elements, message headers, and/or message body contents, and modify a message's recipients, headers, and body. Possible uses for filters include spam rejection, virus filtering, and content control. In general, Milter seeks to address site-wide filtering concerns in a scalable way. Originally designed for sendmail, Milters written for sendmail can now be used with Messaging Server, although some limitations may apply (see below). For more information on Milter refer to the Internet.

Milter/Messaging Server Theory of Operations

Milters control what action is to be performed on a message. Messaging Server controls what messages are to be acted upon by a Milter using the methods described in Specifying the Messages to Be Filtered.

In sendmail, Milter consists of support code in sendmail itself and a separate `libmilter` library. Filter authors link their filters against `libmilter` to produce a server. Sendmail is then configured to connect to these Milter servers.

Messaging Server provides a library that emulates the sendmail side of the Milter interface. This allows Milters written for sendmail to be used with Messaging Server.

A few words of caution are in order here. The Milter protocol consists of a complex mixture of text and binary elements and is not well documented. Additionally, Milter semantics are tightly coupled with sendmail's way of processing messages. In particular, Milters can and usually do access a subset of the macros defined in the sendmail configuration. Messaging Server's Milter client library attempts to provide a reasonable set of sendmail macros, but it is entirely possible for a Milter to be written that depends on a specific aspect of sendmail configuration which is not currently implemented. The net result is that an arbitrary Milter pulled off the network may or may not work with this client library. If problems develop we'll try and resolve them, but we cannot possibly guarantee success with every Milter out there.

Milter Requirements and Usage Considerations

The Milter server can run on a separate system of its own, on the same system as Messaging Server, in a single system deployment, or, in a two-tier deployment, on the same system as the MTA. When LMTP is used between the MTA and the message store, the filtering must be invoked from the MTA, it can not be invoked from the message store. When SMTP is used between the MTA and the message store it can be invoked from either one, and it can run on either system or a third separate system.

Messaging Server supports connecting to multiple Milter servers. If you specify a domain name that translates to multiple IP addresses, the system will try all of them in order received from the DNS until one works. Some DNS servers support randomizing the order of addresses they return, so this provides a primitive load balancing/failover facility.

Supported Milter Capabilities and Actions

The Milter interface currently supports several capabilities and actions. The following table shows the status of support for the Milter capabilities, their descriptions, and the version they were added to Messaging Server.

Capability	Description	Version Support Was Added
SMFIP_NOCONNECT	Skip SMFIC_CONNECT	6.3
SMFIP_NOHELO	Skip SMFIC_HELO	6.3
SMFIP_NOMAIL	Skip SMFIC_MAIL	6.3
SMFIP_NORCPT	Skip SMFIC_RCPT	6.3
SMFIP_NOBODY	Skip SMFIC_BODY	6.3
SMFIP_NOHDRS	Skip SMFIC_HEADER	6.3
SMFIP_NOEOH	Skip SMFIC_EOH	6.3
SMFIP_NR_HDR	Skip SMFIC_HEADER responses	7U4
SMFIP_NOUNKNOWN	Skip unknown commands	8.0
SMFIP_NODATA	Skip SMFIC_DATA	8.0
SMFIP_SKIP	MTA understands SMFIS_SKIP	7U4
SMFIP_RCPT_REJ	MTA should also send rejected RCPTs	8.0
SMFIP_NR_CONN	No reply for connect	8.0
SMFIP_NR_HELO	No reply for HELO	8.0
SMFIP_NR_MAIL	No reply for MAIL	8.0
SMFIP_NR_RCPT	No reply for RCPT	8.0
SMFIP_NR_DATA	No reply for DATA	8.0
SMFIP_NR_UNKN	No reply for UNKN	8.0
SMFIP_NR_EOH	No reply for end of header	8.0
SMFIP_NR_BODY	No reply for body chunk	8.0
SMFIP_HDR_LEADSPC	Header value leading space	8.0

The following table shows the status of support for the Milter actions, their descriptions, and the version they were added to Messaging Server.

Action	Description	Version Support Was Added
SMFIF_ADDHDRS	Add headers	6.3
SMFIF_CHGBODY	Change body chunks	6.3
SMFIF_ADDRCPT	Add recipients	6.3
SMFIF_DELRCPT	Remove recipients	6.3
SMFIF_CHGHDRS	Change or delete headers	6.3
SMFIF_QUARANTINE	Quarantine message	6.3
SMFIF_CHGFROM	Filter may change from	8.0
SMFIF_ADDRCPT_PAR	Add recipients including args	unsupported
SMFIF_SETSYMLIST	Can send set of wanted macros	unsupported

Macros Provided by the Milter Interface

The following table shows the macros currently defined by the Milter interface and their descriptions:

Macro	Description
${auth_authen} (MAIL FROM)	Authenticated sender address.
${auth_author} (MAIL FROM)	The value of the AUTH parameter to MAIL FROM.
${client_addr} (CONNECT)	The IP address of the SMTP client, expressed as a dotted quad value. Only set when SMTP over TCP is being used.
${destination_channel} (RCPT TO)	MTA destination channel for the current recipient.
$i (MAIL FROM)	Queue ID for the current message. The MTA generates a unique ID for each session. This ID is what appears in the $i macro.
$j (CONNECT)	Text placed in the "by" clause of Received: header fields. This is controlled by the RECEIVED_DOMAIN MTA option. If the option is not set the official host on the local channel is used instead.
${mail_addr} (MAIL FROM)	The MAIL FROM address for the current transaction.
${mail_host} (MAIL FROM)	The host part of the MAIL FROM address for the current transaction.
${optin} (RCPT TO)	Spamfilter optin value for the current RCPT TO address.
${rcpt_addr} (RCPT TO)	Current RCPT TO address.
${rcpt_host} (RCPT TO)	The host part of the current RCPT TO address.
${rcpt_mailer} (RCPT TO)	Set to "local" for valid recipient addresses, "error" for invalid addresses.
${source_channel} (MAIL FROM)	MTA source channel.

The smfi_insheader modification action associated with the SMFIF_ADDHDRS action flag specifies an index into the header where the field is to be inserted. Such semantics are not provided by Sieve; indeed, the smfi_insheader documentation itself notes that indices are not reliable. Prior to Messaging Server 8.0, smfi_insheader was implemented by using a plain Sieve addheader for an index of 0 and addheader :last for a nonzero index. However, some milters, notably OpenDKIM, have been observed using an index value of 1 in an attempt to insert a field above the Received: field added by sendmail. Accordingly a new Milter spamfilter option has been added to deal with this and similar situations:

MAX_PREPEND_INDEX (integer, default = 1)

MAX_PREPEND_INDEX specifies the smallest index value that can be passed to smfi_insheader by the Milter server and cause the resulting header field to be inserted at the top of the header block rather than the bottom.

The libmilter provided by sendmail 8.14 now supports Milter session reuse for multiple SMTP sessions or transactions. Unfortunately this support does not appear to be negotiated, making it necessary to have an option to enable it in addition to various options to control its use. Accordingly, several new options have been added to the milter spamfilter option file. The following table lists these options and their descriptions.

Milter spamfilter Option File Option	Description
USE_QUIT_NC (boolean)	Enable use of the QUIT_NC milter command so sessions can be reused. This should only be set when the version of libmilter is recent enough to support the feature. (Sendmail 8.14 or later.) Default: 0
SESSION_INACTIVITY_TIMEOUT (integer)	Time in session a session is allowed to remain idle and still be a candidate for reuse. Default: 180
TRANSACTIONS_PER_SESSION (integer)	Number of transactions allowed in a single session. Default: 100
SESSION_TIME (integer)	Maximum time, in seconds, that a single session can be used. Default: 3600

Starting with Messaging Server 8.0, proper remote port information is transferred through the smfi_connect callback.

Finally, support for milter connections via Socks has been removed in Messaging Server 8.0, so the SOCKS_HOST, SOCKS_PORT, SOCKS_USERNAME, and SOCKS_PASSWORD Milter options are no longer supported.

Per-recipient Modification Actions

A Milter extension to allow per-recipient modification actions has been implemented. The extension is enabled by setting the PER_RECIPIENT_ACTIONS option to 1 in the Milter plugin options file. Since it's possible that this extension may conflict with someone else's private extension, this option defaults to 0.

This extension consists of a new SMFIR_SPECRCPT action Milters can send to the MTA. This action specifies one or more recipient addresses. After this action is sent any subsequent modification actions will only apply to the specified recipient addresses. Additional SMFIR_SPECRCPT actions can be sent in order to specify additional sets of recipient addresses.

Currently, the modification actions supported by the extension are:

SMFIR_ADDHEADER Add header
SMFIR_INSHEADER Insert header
SMFIR_CHGHEADER Change header
SMFIR_QUARANTINE Quarantine message
SMFIR_CHGFROM Change envelope sender (from)

The following supported modification actions don't make sense in a recipient-specific context:

SMFIR_ADDRCPT Add recipient
SMFIR_DELRCPT Remove recipient

Nothing prevents the Milter from sending these modification actions in a recipient-specific context, but they won't have a recipient-specific effect.

Finally, the modification action:

SMFIR_REPLBODY Replace body

is too expensive in a recipient-specific context and is therefore not supported. The effect of specifying it is undefined.

Note that since this is an Oracle extension, sendmail's libmilter library does not provide any support. This can be implemented by making the following modifications to libmilter:

1. Add the following constant and routine declaration to

include/libmilter/mfapi.h:

```
/\* Oracle extension for recipient-specific modification actions \*/
\#define SMFIF_SPECRCPT 0x1000000 // Recipient-specific modification
actions

/\*
** Specify a recipient for whom subsequent modification actions
** apply. Modification actions specified prior to this point will
** no longer apply to this recipient.
\*\*
** SMFICTX \*ctx; Opaque context structure
** char \*rcpt; Null terminated list of null terminated envelope
** recipient addresses subsequent actions apply to. These should
** be in exactly the form passed to xxfi_envrcpt or the address
** may not be selected for subsequent modification actions.
\*/

LIBMILTER_API int smfi_specrcpt \__P((SMFICTX \*, char \*));
```

2. Add the following constant to include/libmilter/mfdef.h:

```
/\* Oracle extension for recipient-specific modification actions \*/
\#define SMFIR_SPECRCPT 'v' /\* Per-recipient modification actions \*/
```

3. Add the following routine to libmilter/smfi.c:

```
/\*
** SMFI_SPECRCPT -- Recipient-specific result (Oracle extension)
\*\*
** Parameters:
** ctx -- Opaque context structure
** rcpt -- null terminated list of null terminated
** recipient addresses
\*\*
** Returns:
** MI_SUCCESS/MI_FAILURE
\*/

int
smfi_specrcpt(ctx, rcpt)
SMFICTX \*ctx;
char \*rcpt;
{
size_t len, l;
struct timeval timeout;
char \*ptr;

if (rcpt == NULL \|\| \*rcpt == '\0')
return MI_FAILURE;
if (\!mi_sendok(ctx, SMFIF_SPECRCPT))
return MI_FAILURE;
timeout.tv_sec = ctx->ctx_timeout;
timeout.tv_usec = 0;
len = 0;
ptr = rcpt;
do
{
len += (l = strlen(ptr) + 1);
ptr += l;
} while (*ptr \!= '\0');
return mi_wr_cmd(ctx->ctx_sd, &timeout, SMFIR_SPECRCPT, rcpt, len);
}
```

With these changes in place recipient-specific handling is enabled by negotiating `SMFIF_SPECRCPT` and activated by calling `smfi_specrcpt` from the Milter eom routine.

To Deploy Milter

Perform the following steps to deploy Milter.

1. Obtain and configure a Milter that will perform the actions you desire.
   Refer to specific Milter documentation for obtaining and configuring information.
2. Load and configure the Milter client library. (See Loading and Configuring the Spam Filtering Software Client Library.)
   a. Specify the path to the client library, `libmilter.so`. Specify the path and name of the Milter configuration file. Example:

```
spamfilter1_library=/opt/sun/comms/messaging/lib/libmilter.so
spamfilter1_config_file=/opt/sun/comms/messaging/lib/milter.opt
```

b. Create a Milter configuration file with the desired options as defined in Milter Options.

3. Specify what messages to send to Milter.
   Messages can be filtered by user, domain, or channel. See Specifying the Messages to Be Filtered.
4. Set the `spamfilter`*X*`_string_action` option in the `option.dat` file:
   `spamfilter`*X*`_string_action=data:,$M`
   This setting is used unconditionally, but must be in the MTA options file for Milter to work properly.

Milter Options

The Milter option file consists of lines of the form `option=value`. The two required options are `HOST` and `PORT`. `HOST` must be set to the name of the system where the Milter server is running while `PORT` must be set to the port the Milter server is configured to listen on. Note that only TCP/IP connections are supported; UNIX domain sockets cannot be specified or used.

Further additional options are available for this options file are shown below.

Milter Options

Option	Description	Default
`DEBUG`	Integer. Enables or disables debug output from the Milter client library. The larger the value, the more debugging output will be produced. 0 produces no output. 1 provides basic debugging. 2 adds logging of TCP traffic. (Debug output from the Milter server is typically controlled by a setting on the command line used to start the server. Note that most Milters seem to only provide the ability to direct debug output to syslog.)	0
`PRESERVE_BREAKS`	Integer. Setting it to 1 will cause CRLFs to map to LF, 0 does the usual header unfolding.	0
`RESETDEBUG`	Integer. Enables per-session debugging when channel debugging is also enabled. While you can't enable Milter debugging independently from a mapping, you can add it onto channel debugging, which can be enabled from, for example, the FROM_ACCESS mapping.	0
`TIMEOUT`	Integer. Specifies the timeout in seconds for operations involving the Milter connection. Available in MS 6.3 and later versions.	3600
`USE_JETTISON`	(New in MS7u2) Integer. If this option is set to 1 sieve jettison will be used instead of discard in the sieves if the milter calls for the message to be discarded. A value of 0 causes discard to be used.	0

Cloudmark Anti-Abuse Client

Messaging Server also works with the Cloudmark Anti-Abuse Client. See the Cloudmark documentation for more information.

Other Anti-Spam and Denial-of-Service Technologies

Adding spam and virus filtering software to the system is the most effective way of reducing spam and viri from getting into user's mailboxes. However, Messaging Server provides a number of other techniques and methods to support spam filtering. These technologies are often used for purposes other than just spam filtering and so are spread throughout this book. Listed below are some sections describing anti-spam and denial-of-service technologies.

Anti-Spam Technologies:

- Anti-Spam Technique: Delay Sending the SMTP Banner
- Routing After Address Validation But Before Expansion
- Handling Forged Email by Using the Sender Policy Framework
- Mail Filtering and Access Control
- Configuring Client Access to POP, IMAP, and HTTP Services
- DNS Domain Verification
- Expansion of Multiple Addresses
- To Create MTA-Wide Filters
- Configuring SMTP Relay Blocking

Denial of Service Technologies:

- Throttling Incoming Connections Using MeterMaid
- Monitoring the Size of the Message Queues
- Monitoring Inbound SMTP Connections

Anti-Spam Technique: Delay Sending the SMTP Banner

A useful spam-fighting strategy is to delay sending the SMTP banner for a brief time (half a second, say), then clear the input buffer, and finally send the banner. The reason this works is that many spam clients are not standards-compliant and starts spewing SMTP commands as soon as the connection is open, ignoring any responses the server sends. Spam clients that do this when this capability is enabled will lose the first few commands in the SMTP dialogue, rendering the remainder of the dialogue invalid.

This feature has now been implemented in Messaging Server. It can be enabled unconditionally by setting the `BANNER_PURGE_DELAY` SMTP channel option to the number of centiseconds to delay before purging and sending the banner. A value of 0 disabled both the delay and purge.

The `PORT_ACCESS` mapping can also be used to control this capability. Specifying `$D` in the template causes an additional argument to be read from the template result after the mandatory SMTP `auth rulset` and realm, and optional application information addition. This value must be an integer with the same semantics as the `BANNER_PURGE_DELAY` value. Note that any `PORT_ACCESS` mapping setting overrides the `BANNER_PURGE_DELAY` SMTP channel option.

Chapter 16. Handling Forged Email by Using the Sender Policy Framework

Handling Forged Email by Using the Sender Policy Framework

Spam producers and email scammers often forge email by using false domain names and email addresses or by using legitimate domain names and email addresses to fool users into thinking that a message is from someone or some company they know. For example, a spammer could send email from an address such as `president@whitehouse.gov` and the user could be fooled into thinking the mail was actually from this address. Forging email might fool users into opening the unsolicited message, or worse, provide information to a false authority. Also, spammers prefer to send their email from legitimate domains that are not on an RBL list.

Sender Policy Framework (SPF) is a technology that can detect and reject forged email during the SMTP dialogue. Specifically, SPF is a protocol that allows a domain to explicitly authorize the hosts that may use its domain name. In addition, a receiving host may be configured to check this authorization. SPF can thus significantly reduce the instances of forged email.

Support for Sender Policy Framework was introduced in **Messaging Server 6.3**.

Note
When using Unified Configuration, you use the `msconfig` command to configure options instead of editing the legacy configuration files.

Topics:

Theory of Operations

When a message comes into Messaging Server, the MTA does an SPF query to determine if the address actually came from the domain on the address. An SPF query consults the DNS for TXT records belonging to the domain of the message (domain). Domain is either the domain name specified as the argument for HELO or EHLO (if the `spfhelo` channel keyword is used) or the domain name in the originator's address given in the MAIL FROM: command (typically the part after the @ character). If no domain name is specified or available, the one specified during HELO/EHLO is used as domain. Most ISPs distribute an authorized list of IP addresses that match their domains. If the IP address does not match the domain name, then the message is assumed to be forged.

Note
Prior to querying the DNS, the software checks the SPF_LOCAL mapping table for a match for domain. If a match is found there, it will be used first.

If a record found from the mapping table contains a `redirect=` domain clause, then the redirection to domain will be done as a DNS query, skipping the recursive and redundant mapping file check.

An example of a resulting TXT record is:

```
v=spf1 +mx a:colo.siroe.com/28 -all
```

The `v=spf1` token is required for SPF records supported by this RFC.

`+mx` checks MX records for domain and confirms that the source IP address for this SMTP connection matches one of the IP addresses given as a result of an MX query for domain. If there is a match, the + means that the result of this is `Pass`.

`a:colo.siroe.com/28` checks for A records for `colo.siroe.com`, then confirm that the source IP address for this SMTP connection is in the same specified CIDR subnet as the A records, comparing only 28 bits (masked against 255.255.255.240). No qualifier character was specified, so it defaults to + meaning that a match results in a `Pass`.

Finally, `-all` matches everything else and results in `Fail`. For a complete description of SPF records, refer to RFC 4408 at http://www.ietf.org/rfc/rfc4408.txt.

SPF processing can have one of several results. The following table shows the results and their descriptions.

SPF Processing Results

Result	Description
`Pass`	The lookup passed, meaning that an SPF record was found and the record validated the originating system as being authorized to use domain.
`Fail`	The lookup found a matching SPF record, however, the record explicitly denied authorization for the SMTP client to use domain during the SMTP transaction. The default behavior of Messaging Server's SPF implementation is to reject the SMTP command with a 5xx reply.
`SoftFail`	The lookup found a matching SPF record, and the record also denies authorization for the SMTP client to use domain. However, the denial is less strict and the record does not direct an outright failure. The default behavior of Messaging Server's implementation is to accept the message, but note the `SoftFail` in the Received-SPF: header for subsequent evaluation such as during Sieve processing.
`Neutral`	The SPF record makes no claim to the SMTP client's authorization to use domain. The message will be accepted. The specification requires that `Neutral` be treated the same as `None`.
`None`	No matching SPF record was found, therefore no SPF processing was done.
`PermError`	A permanent error was encountered during SPF processing, such as syntax errors in the SPF record, DNS failures while processing nested SPF records (due to `include:` mechanism or a `redirect=` modifier), or exceeding configured limits for SPF processing while processing nested SPF records. The default behavior is to reject the SMTP command with a 5xx reply.
`TempError`	A temporary error was encountered during SPF processing, most likely due to DNS timeouts querying SPF records. The default behavior is to reject the SMTP command with a 4xx reply.

After SPF processing has completed, a `Received-SPF:` header is written to the message documenting the result of the SPF processing. This header can then be queried during Sieve processing for subsequent consideration. Extensive debugging is available if the MTA option `MM_DEBUG` is enabled (>0). In Unified Configuration, run the `msconfig` command to set the option. In legacy configuration, set the option in the `option.dat` file.

SPF Limitations

SPF is only one tool to use to fight spam, and it does not address all issues. A spammer can easily create a domain and add an SPF TXT record that makes the domain seem legitimate. On the other hand, SPF is very effective for detecting forged email from established ISPs, although many TXT records allow the SPF to not fail.

SPF Pre-Deployment Considerations

It is important to have a very fast DNS server on your system because a DNS query for every message is required.

Setting up the Technology

The two steps to set up SPF technology are:

- Place channel keywords on the incoming TCP channel (typically the `tcp_local` channel, although there might be other channels if you allow channel switching from `tcp_local` to another channel). See SPF Keywords for more information.

- Set up the options. In Unified Configuration, run the `msconfig` command. In legacy configuration, edit the `option.dat` file. See SPF Limiting Options for more information.

Reference Information

This section provides reference information for the SPF channel keywords and the SPF MTA options. SPF support is implemented through four channel keywords applied to the incoming `tcp_*` channel (typically `tcp_local`). The following table shows the keywords and their descriptions. In Unified Configuration, use the `msconfig edit channels` command to edit SPF channel keywords. In legacy configuration, edit the `imta.cnf` file.

SPF Keywords

Keyword	Description
`spfnone`	Disables SPF processing
`spfhelo`	Enables SPF processing for the domain name specified as an argument to HELO or EHLO.
`spfmailfrom`	Enables SPF processing for the domain name provided for the originator envelope address after receiving the MAIL FROM:.
`spfrcptto`	Enables SPF process for the domain name provided for the originator envelope address after receiving the RCPT TO:. Processing is the same as `spfmailfrom` except that it is delayed in the SMTP transaction until after the RCPT TO: command has been issued and the recipient has otherwise been confirmed to be a valid recipient.

`spfmailfrom` and `spfrcptto` are conflicting keywords and you should only specify one of these two keywords on the channel. You can, however, use `spfhelo` in conjunction with either `spfmailfrom` or `spfrcptto` to perform both kinds of SPF checks.

Additional support to establish limits on SPF processing and to control whether SMTP commands will be accepted, failed with a 4xx response (temporary failure), or failed with a 5xx response (permanent failure) for the various SPF results includes: `Fail`, `SoftFail`, `PermError`, and `TempError`.

Use the following MTA options to place limits on SPF processing. In Unified Configuration, set options with the `msconfig` command. In legacy configuration, edit the `option.dat` file.

SPF Limiting Options

Option	Description
SPF_MAX_RECURSION	Specifies the number of recursions that will be allowed into nested SPF records due to include: or redirect=. Exceeding this limit will result in a PermError. Default: 10 (mandated by the RFC)
SPF_MAX_DNS_QUERIES	Specifies the number of mechanisms or modifiers that require DNS lookups (including include:, a:, mx:, ptr:, exists:, redirect=, and exp=). The limit is not counted as the number of actual DNS lookups, so one mechanism could lead to several DNS queries. Exceeding this limit will result in a PermError. Default: 10 (mandated by the RFC)
SPF_MAX_TIME	Specifies the number of seconds that will be allowed for the SPF processing to complete. Exceeding this value will result in a TempError. The default value is more generous than the RFC suggests. Default: 45

Additionally, the following MTA options can be configured to control the behavior of the SMTP server in response to SPF results of Fail, SoftFail, PermError, and TempError. In Unified Configuration, run the msconfig command. In legacy configuration, edit the option.datt file. For each of these results, the SMTP server can send back a 2xx (success) response, 4xx (temporary failure), or 5xx (permanent failure). Also, for Fail and SoftFail, the MTA can distinguish between an SPF result as the result of an "all" mechanism versus an otherwise explicitly referenced match. You can then make a distinction between a particular result and the SPF record's default result. The valid values for any of these options is 2, 4, or 5. The values of 2, 4, or 5 correspond to 2xx, 4xx, or 5xx responses from the SMTP server as a result of getting that particular SPF status. So, for example, if SPF_SMTP_STATUS_FAIL=2 and the SPF record explicitly blocks us with a "-a:192.168.1.44" (our IP address), then instead of responding with a 5xx response, we'll accept the address with a "250 OK" instead.

SPF Failure and Error Options

Option	Description
SPF_SMTP_STATUS_FAIL	Used when the match of an SPF record is a "-" flagged mechanism other than "-all." Default: 5
SPF_SMTP_STATUS_FAIL_ALL	Used when the matching mechanism is "-all." Default: 5
SPF_SMTP_STATUS_SOFTFAIL	Used when the match of an SPF record is a "~" flagged mechanism other than "~all." Default: 2
SPF_SMTP_STATUS_SOFTFAIL_ALL	Used when the matching mechanism is "~all." Default: 2
SPF_SMTP_STATUS_TEMPERROR	Used when there is a temporary failure, usually related to DNS processing problems. Default: 4
SPF_SMTP_STATUS_PERMERROR	Used when there is a permanent failure, usually due to syntax or other technical errors found during SPF processing. (This is due to a non-local error.) Default: 5

Testing SPF by Using spfquery

You can use the `spfquery` testing utility to test SPF processing.

> **Note**
> `spfquery` does not test your SPF configuration. It tests what would be returned if you were to enable SPF processing.

Requirements: Must be run as a user who has access to run the Messaging Server binaries and access its libraries such as `root` or `mailsrv`, for example.

Location: *msg-svr-base*`/sbin/`

Syntax

`spfquery` [`-i` *ip-address*] [`-s` *sender-email*] [`-h` *helo-domain*] [`-e none` | `neutral` | `pass` | `fail` | `temperror` | `permerror`] [`-v`] [`-V`] [?] `domain`

The following table shows the `spfquery` options and their descriptions.

spfquery options

Option	Description
`-i` *ip address*	Specifies the IP address to be used as the remote address for the SPF query. Default is `127.0.0.1`. This option can also be `--ip-address`.
`-s` *domain*	The email address that will be used as if it were specified as `MAIL FROM:`. Default: `postmaster@domain`. This option can also be `--sender`.
`-h` *helo-domain*	The domain name as if it were specified for the HELO domain. This domain is not verified itself, but instead provided as supplemental information for macro processing. Default value is the same as the value you specified for domain. This option can also be `--helo-domain`.
`-e` *result*	`spfquery` compares the result of the SPF processing with what is expected and if the result is different, a message is printed and `spfquery` exits with a non-zero return status. Result can be one of: `none`, `neutral`, `pass`, `fail`, `softfail`, `temperror`, or `permerror`. This option can also be `--expect`.
`-v`	Enables verbose output during SPF processing. This option can also be `--verbose`.
`-V`	Prints the current version of the SPF library. This option can also be `--version`.
`-?`	Prints this usage information. This option can also be `--help`.

Example with Debugging Enabled

```
# /opt/SUNWmsgsr/sbin/spfquery -v -i 192.168.1.3 11.spf1-test.siroe.com
    Running SPF query with:
      IP address: 192.168.1.3
          Domain: 11.spf1-test.siroe.com
          Sender: postmaster@11.spf1-test.siroe.com (local-part:
postmaster)
     HELO Domain: 11.spf1-test.siroe.com

    15:30:04.33:
----------------------------------------------------------------
    15:30:04.33: SPFcheck_host called:
    15:30:04.33:       source ip = 192.168.1.3
    15:30:04.33:          domain = 11.spf1-test.siroe.com
    15:30:04.33:          sender = postmaster@11.spf1-test.siroe.com
    15:30:04.33:      local_part = postmaster
    15:30:04.33:     helo_domain = 11.spf1-test.siroe.com
    15:30:04.33:
    15:30:04.33:   Looking up "v=spf1" records for
11.spf1-test.siroe.com
    15:30:04.35:     DNS query status: Pass
    15:30:04.35:       "v=spf1 mx:spf1-test.siroe.com
-all"
    15:30:04.35:
    15:30:04.35:   Parsing mechanism: " mx : spf1-test.siroe.com"
    15:30:04.35:     Assuming a Pass prefix
    15:30:04.35:     Processing macros in spf1-test.siroe.com
    15:30:04.35:     Comparing against 192.168.1.3
    15:30:04.35:     Looking for MX records for spf1-test.siroe.com
    15:30:04.41:       mx02.spf1-test.siroe.com:
    15:30:04.41:         192.0.2.22 - No match
    15:30:04.41:         192.0.2.21 - No match
    15:30:04.41:         192.0.2.20 - No match
    15:30:04.41:         192.0.2.23 - No match
    15:30:04.41:       mx01.spf1-test.siroe.com:
    15:30:04.42:         192.0.2.13 - No match
    15:30:04.42:         192.0.2.11 - No match
    15:30:04.42:         192.0.2.12 - No match
    15:30:04.42:         192.0.2.10 - No match
    15:30:04.42:       mx03.spf1-test.siroe.com:
    15:30:04.42:         192.0.2.32 - No match
    15:30:04.42:         192.0.2.30 - No match
    15:30:04.42:         192.0.2.31 - No match
    15:30:04.42:         192.168.1.3 - Matched
    15:30:04.42:   Mechanism matched; returning Pass
    15:30:04.42:
    15:30:04.42:   Parsing mechanism: "- all : " (not evaluated)
    15:30:04.42:
    15:30:04.42: SPFcheck_host is returning Pass
    15:30:04.42:
----------------------------------------------------------------
```

Handling Forwarded Mail in SPF by Using the Sender Rewriting Scheme (SRS)

As described above, SPF is a mechanism that attempts to prevent email forgery by looking up special TXT records associated with the domain in the mail FROM: (envelope from) address. This operation, which can actually involve several DNS lookups, eventually produces a list of IP addresses that are authorized to send mail from the domain. The IP address of the SMTP client is checked against this list and if it isn't found, the message can be considered to be fraudulent.

SPF presents serious problems for sites that provide mail forwarding services such as universities (for their alumni) or professional organizations (for their members). A forwarder ends up sending out mail from essentially arbitrary senders, which can include senders who have implemented SPF policies and which, of course, do not list the IP addresses of the forwarding system or systems as being permitted to use addresses from their domain.

The Sender Rewriting Scheme, or SRS, provides a solution to this problem. SRS works by encapsulating the original sender's address inside a new address using the forwarder's own domain. Only the forwarder's own domain is exposed for purposes of SPF checks. When the address is used it routes the mail (usually a notification) to the forwarder, which removes the address encapsulation and sends the message on to the real destination.

Of course address encapsulation isn't exactly new. Source routes were defined in RFC 822 and provide exactly this sort of functionality, as does percent hack routing and bang paths. However, these mechanisms are all problematic on today's Internet since allowing their use effectively turns your system into an open relay.

SRS deals with this problem by adding a keyed hash and a timestamp to the encapsulation format. The address is only valid for some period of time, after which it cannot be used. The hash prevents modification of either the timestamp or the encapsulated address.

SRS also provides a mechanism for handling multi-hop forwarding without undue growth in address length. For this to work certain aspects of SRS address formatting have to be done in the same way across all systems implementing SRS.

> Support for Sender Rewriting Scheme was introduced in **Messaging Server 6.3 P1**.

The following MTA options have been added:

- SRS_DOMAIN. This must be set to the domain to use in SRS addresses. Email sent to this domain must always be routed to a system capable of SRS operations for the domain. SRS processing is handled as an overlay on top of normal address processing so nothing prevents a site from using their primary domain as the SRS domain.
- SRS_SECRETS. This is a comma separated list of secret keys used to encode and decode SRS addresses. The first key on the list is used unconditionally for encoding. For decoding, each key is tried in order to generate a different hash value. The decoding operation proceeds if any of the hashes match.
  The ability to use multiple keys makes it possible to change secrets without service disruption: Add a second key, wait for all previously issued addresses to time out, and then remove the first key.
- SRS_MAXAGE. Optionally specifies the number of days before a message times out. The default if the option isn't specified is 14 days.
  Every system that handles email for the selected SRS domain must be configured for SRS processing and must have all three SRS options set identically.

Setting these options is sufficient to enable SRS address decoding. Encoding is another matter and should should only be done to envelope From: addresses you know are associated with forwarding activity. SRS encoding is controlled by six new channel keywords: addresssrs, noaddresssrs, destinationsrs, nodestinationsrs, sourcesrs, and nosourcesrs.

Three conditions have to be met for SRS encoding to occur:

1. The current source channel has to be marked with `sourcesrs`. (`nosourcesrs` is the default).
2. The current destination channel has to be marked with `destinationsrs`. (`nodestinationsrs` is the default).
3. The current address, when rewritten, has to match a channel marked `addresssrs`. (`noaddress` is the default).

Encoding only occurs when all of these conditions are true. The simplest setup consists of pure forwarding where all messages enter and exit on the `tcp_local` channel and all non-local addresses need SRS handling. In such a setup, `tcp_local` would be marked with the three keywords `sourcesrs`, `destinationsrs`, and `addresssrs`.

Finally, `imsimta test -rewrite` has been enhanced to show SRS encoding and decoding results for whatever address is input. For example, the address `foo@example.com` might produce the output similar to:

```
SRS encoding = SRS0=dnG=IS=example.com=foo@example.org
```

If this encoded address is rewritten it produces the following output:

```
SRS decoding = foo@example.com
```

`imsimta test -rewrite` also shows any errors that occur during SRS decoding.

Chapter 17. LMTP Delivery

LMTP Delivery

The Oracle Communications Messaging Server MTA can use LMTP Local Mail Transfer Protocol, defined in RFC 2033) for delivery to the message store in situations where a multi-tier messaging server deployment is used. In these scenarios, where you are using inbound relays and back end message stores, the relays become responsible for address expansion and delivery methods such as autoreply and forwarding and also for mailing list expansion. Delivery to the back end stores historically has been over SMTP which requires the back end system to look up the recipient addresses in the LDAP directory again, thereby engaging the full machinery of the MTA. For speed and efficiency, the MTA can use LMTP rather than SMTP to deliver messages to the back end store. The Messaging Server's LMTP server is not intended as a general purpose LMTP server, but rather as a private protocol between the relays and the back end message stores. For simplicity of discussion, examples involving two-tier deployments are used.

> **Note**
> LMTP is recommended for use in multi-tier deployments. Also, the Messaging Server's LMTP service as implemented is not designed to work with other LMTP servers or other LMTP clients.

Topics:

LMTP Delivery Features

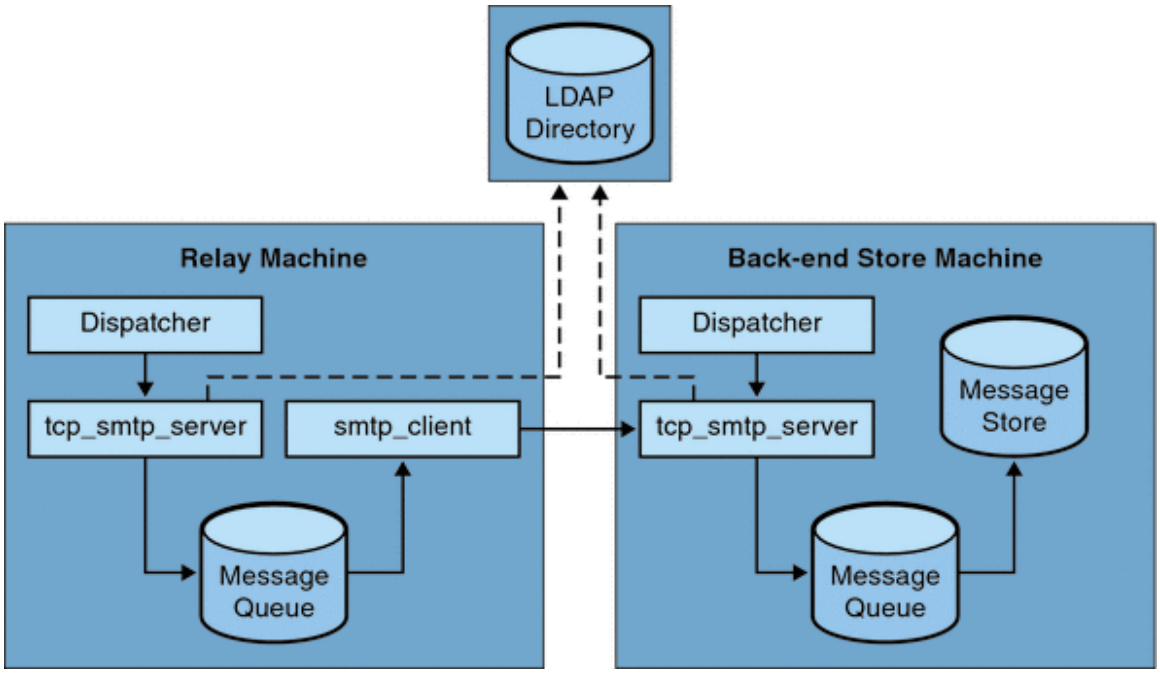
The MTA's LMTP server is more efficient for delivering to the back end message store because it:

- Reduces the load on the back end stores.
  Because relays are horizontally scalable and back end stores are not, it is good practice to push as much processing to the relays as possible.
- Reduces the load on the LDAP servers.
  The LDAP infrastructure is often a limiting factor in large messaging deployments.
- Reduces the number of message queues.
  Having queues on both the relay and the back end store makes finding a lost message that much harder for the administrator of a messaging deployment.

Messaging Processing in a Two-Tier Deployment Without LMTP

Two Tier Deployment Without LMTP presents in pictorial form the following discussion of message processing in a two-tier deployment scenario without LMTP.

Two Tier Deployment Without LMTP



Without LMTP, in a two level deployment with relays in front of the store systems, the processing of an inbound message begins with a connection on the SMTP port picked up by the dispatcher on the relay machine and handed off to a `tcp_smtp_server` process. This process does a number of things with the inbound message including:

- Looking up the user in the directory
- Determining if the user is within a domain hosted by this email deployment
- Determining if the user is a valid user in the domain
- Rewriting the envelope address as `@mailhost:user@domain`
- Enqueuing the message for delivery to the mailhost

The `smtp_client` process then picks up the mail message from the queue and sends it to the mailhost. On the mailhost, some very similar processing takes place. A connection on the SMTP port is picked up by the dispatcher and handed off to a `tcp_smtp_server` process. This process does a number of things to the message, including:

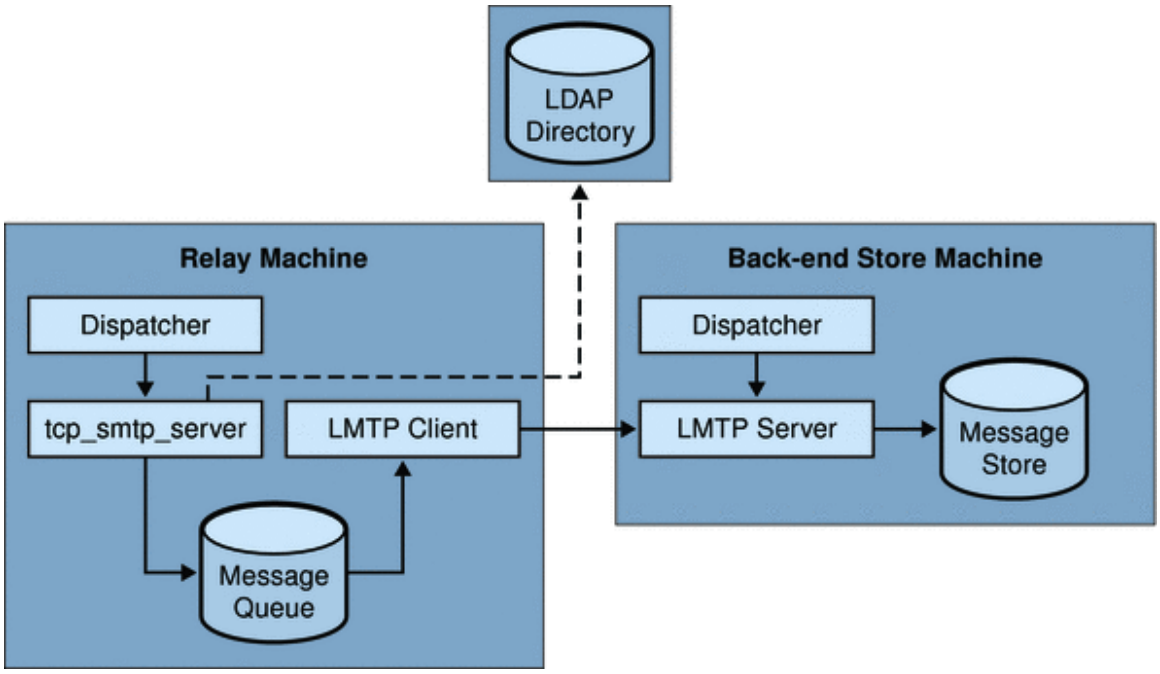
- Looking up the user in the directory
- Determining if the user is within a domain hosted by this email deployment
- Determining if the user is a valid user in the domain
- Rewriting the envelope address to direct the message to the `ims_ms` channel
- Enqueuing the message for delivery to the store

Then the `ims_ms` process picks up the mail message and attempts to deliver it to the store. In this scenario, the enqueuing processing is performed twice, and the MTAs each perform an LDAP lookup.

Messaging Processing in a Two-Tier Deployment With LMTP

Messaging Processing in a Two-Tier Deployment With LMTP presents in pictorial form the following discussion of message processing in a two-tier deployment scenario with LMTP.

Two Tier Deployment With LMTP



With LMTP in place, a connection on the SMTP port of the relay machine is picked up by the dispatcher and handed off to a `tcp_smtp_server` process. This process does a number of things with the inbound message including:

- Looking up the user in the directory
- Determining if the user is within a domain hosted by this email deployment
- Determining if the user is a valid user in the domain
- Determining which back end message store machine hosts the mailbox for the user
- Enqueuing the message for delivery to the mailhost

On the store machine, a connection to the LMTP port is received by the dispatcher and handed off to the `lmtp_server` process. The LMTP server then inserts the message into the user's mailbox or into the UNIX native mailbox. If message delivery is successful, the message is dequeued on the relay machine. If unsuccessful, the message remains on the relay machine. Note that the LMTP process on the message store does not engage any MTA machinery for processing addresses or messages.

LMTP Overview

For the most part, the MTA itself can be basically absent from the back end server. The only necessary MTA components are:

- The dispatcher
- `libimta`
- The LMTP server
- The `imta.cnf` file
- The `mappings` file
- The `imta.tailor` file

While the dispatcher requires MTA configuration files, these files can be very short. The dispatcher must run on the back end server so that it can start the LMTP servers which run under it. Because the dispatcher and the LMTP server use various functions of `libimta`, this needs to be present on the back end server as well.

The LMTP server does not perform any of the usual MTA enqueuing or dequeuing functions, header processing, or address translations. The relay system performs all the manipulation of the content of the messages and addresses which then presents to the LMTP server the message in exactly the form to be

delivered to the message store and with the delivery address already in the form required by the store. Additional recipient information that is usually available as a message is delivered to the store, such as the user's quota, is presented along with the recipient address as LMTP parameters. Should a delivery attempt fail, the message is left enqueued in the LMTP queue on the relay system.

To Configure LMTP Delivery

Configuring the LMTP delivery mechanism requires configuration on both the relay machines and on the back end stores. On the relays, the `DELIVERY_OPTIONS` MTA option (in `option.dat`) has to be changed so that messages being delivered to the stores are passed to the LMTP channel. The back end store must be configured with the dispatcher, but does not need the job controller. The dispatcher must be configured to run the LMTP server.

Tip
Run the following command to get the `option.dat` file defaults:

```
imsimta cnbuild -noimage -option_file=/tmp/x4
```

In a typical multi-tier deployment, users are provisioned on different backend message store machines. One or more of these backend machines may not have LMTP turned on and therefore the front-end relays need to be aware of which store machines are LMTP aware. This is achieved by using the General Database facility to explicitly name those message stores which are configured to accept LMTP delivery.

To Configure the Inbound MTA Relays with LMTP

To configure inbound MTA relays to use LMTP, do the following:

1. Modify your `imta.cnf` file:
   a. Change the LMTP rewrite rules to read:

```
! lmtp
.lmtp   $E$F$U%$H.lmtp@lmtpcs-daemon
.lmtp   $B$F$U%$H@$H@lmtpcs-daemon
!
```

   b. Add the channel keywords `multigate connectcanonical` to each of the `tcp_lmtp*` channel blocks. If the channel block does not exist, add one. If one already exists, then uncomment the information by deleting the "!" at the beginning of the line so that the name of the `tcp_lmtp*` channel and the official host name are flush against the left margin.
   c. Add the following channel keywords to the `tcp_lmtpcs` channel:

```
fileinto @$4O:$U+$S@$D
```

   Note that the 'O' in the keyword above is a capital letter O, not a zero.
   d. After your changes the modified `imta.cnf` rewrite rules should look like this:

```
! lmtp
.lmtp   $E$F$U%$H.lmtp@lmtpcs-daemon
.lmtp   $B$F$U%$H@$H@lmtpcs-daemon
!
```

The changed channel blocks should look like this:

```
!
! tcp_lmtpcs (LMTP client - store)
tcp_lmtpcs defragment lmtp multigate connectcanonical fileinto \
   @$4O:$U+$S@$D port 225 nomx single_sys pool SMTP_POOL \
  dequeue_removeroute
lmtpcs-daemon
```

2. Set the mailbox `DELIVERY_OPTIONS` in the `option.dat` file to:

```
#*mailbox=@$X.LMTP:$M%$\$2I$_+$2S@lmtpcs-daemon
```

The incoming MTA relay configuration settings should look like this:
The `option.dat` entry for `DELIVERY_OPTIONS` should look like this:

```
!-----------------------------------------
! Modified DELIVERY_OPTIONS to activate LMTP
! delivery from a frontend to the backend store
!-----------------------------------------
!
DELIVERY_OPTIONS=\
    #*mailbox=@$X.LMTP:$M%$\$2I$_+$2S@lmtpcs-daemon,\
    #&members=*,\
    #&@members_offline=*,\
    #/hold=@hold-daemon:$A,\
    #program=$M%$P@pipe-daemon,\
    #forward=**,\
    #*^!autoreply=$M+$D@bitbucket
!
```

Note
You need to add the full `DELIVERY_OPTIONS` as shown in the preceding example. There is no way to override just the default for one option. If you specify `DELIVERY_OPTIONS` at all, you must define them all.

To Configure Back End Stores with LMTP and a Minimal MTA

The back end stores require only a minimal MTA if they are receiving messages over LMTP. They require a dispatcher, a job controller and a simple MTA configuration. In particular they need a `dispatcher.cnf`, `job_controller.cnf` and a `mappings` file which comprise the only significant part of the MTA configuration.

The `dispatcher.cnf` file must contain the following:

```
! VERSION=1.1
! IMTA default dispatcher configuration file
!
! Global defaults
!
MIN_PROCS=1
MAX_PROCS=10
MIN_CONNS=30
MAX_CONNS=50
MAX_SHUTDOWN=2
MAX_LIFE_TIME=86400
MAX_LIFE_CONNS=10000
MAX_IDLE_TIME=600
HISTORICAL_TIME=0
!
! rfc 2033 LMTP server - store
!
[SERVICE=LMTPSS]
PORT=225
IMAGE=IMTA_BIN:tcp_lmtp_server
LOGFILE=IMTA_LOG:tcp_lmtpss_server.log
PARAMETER=CHANNEL=tcp_lmtpss
STACKSIZE=2048000
! Uncomment the following line and set INTERFACE_ADDRESS to an
! appropriate host IP (dotted quad) if the dispatcher needs to
! listen on a specific interface (e.g. in a HA environment).
! INTERFACE_ADDRESS=!
! rfc 2033 LMTP server - native
!
```

Note that by default, the LMTP services in the `dispatcher.cnf` file are commented out. You must uncomment them to get LMTP to work.

The normal dispatcher options of `MAX_CONNS`, `MAX_PROCS`, `MAX_LIFE_CONNS`, and `MAX_LIFE_TIME` can also be set, but need to be set appropriately for your hardware.

The `PORT_ACCESS` mapping is important. The LMTP implementation for the back-end servers is intended as a private protocol between Messaging Server relays and back end stores. You must use the `PORT_ACCESS` mapping to make sure that only such relays and the server itself can connect to these services. (The server needs to be able to connect to itself for monitoring purposes.) Your mapping file should look something like this:

```
PORT_ACCESS

  TCP|*|225|192.18.74.206|* $Y
  TCP|*|225|192.18.74.129|* $Y
  TCP|*|225|127.0.0.1|* $Y
  TCP|*|*|*|*   $N500$ Do$ not$ connect$ to$ this$ machine
```

These IP addresses are LMTP server and client IP address. You should replace the sample IP addresses specified in the `PORT_ACCESS` mapping table here with the IP addresses of your relays on the network that connect to the back-end stores.

There has to be an `imta.cnf` file, but it is there merely to make the configuration complete. A minimal

imta.cnf file consists of the following channel definitions:

```
!
! IMTA configuration file
!
! tcp_lmtpss (LMTP server - store)
tcp_lmtpss lmtp flagtransfer
tcp_lmtpss-daemon
```

Note that by default, the LMTP channel definitions are commented out. You must uncomment them if you want LMTP to work.

You can use the default job_controller.cnf file created on installation. No modification of this file is required.

Configuring Relays for Sending Messages Via LMTP to Back End Systems with Message Stores and Full MTAs

There are situations where you might want the back end stores to have the full capabilities of the MTA but still to have the load savings of using LMTP. For example, you might want program delivery on the back end store. In this case the relays should be configured as described above in To Configure the Inbound MTA Relays with LMTP.

Configuring LMTP on Back End Message Store Systems Having Full MTAs

The only changes from the configuration of a back end store messaging system to one with LMTP direct delivery to the store are that the following lines need to be added to the end of the dispatcher.cnf file:

```
! rfc 2033 LMTP server - store
[SERVICE=LMTPSS]
PORT=225
IMAGE=IMTA_BIN:tcp_lmtp_server
LOGFILE=IMTA_LOG:tcp_lmtpss_server.log
PARAMETER=CHANNEL=tcp_lmtpss
STACKSIZE=2048000
! Uncomment the following line and set INTERFACE_ADDRESS to an
! appropriate host IP (dotted quad) if the dispatcher needs to
! listen on a specific interface (e.g. in a HA environment).
!INTERFACE_ADDRESS=
```

Note that by default, the LMTP services in the dispatcher.cnf file are commented out. You must uncomment them to get LMTP to work. You need both the dispatcher.cnf file defining what the dispatcher should do with respect to this channel and the channel defined in the imta.cnf file. Also, the LMTP port numbers are just examples, and can be anything you choose.

This is the same as the whole dispatcher.cnf file described above for when the back end store is configured only for LMTP. The mappings file also requires the PORT_ACCESS mappings as described for LMTP only back end stores.

Handling 4.2.1 Mailbox Busy Error in Response to LMTP Message Data

If the LMTP channel option MAILBOX_BUSY_FAST_RETRY is set to 1 (the default) a 4.2.1 Mailbox busy error in response to LMTP message data is handled by retrying the message after a random but short interval; normal message backoff values do not apply. Setting the option to 0 disables this behavior.

LMTP Protocol as Implemented

This section provides a sample LMTP dialogue with an explanation of what is seen in that dialogue. The LMTP client on the relay uses standard LMTP protocol to talk to the LMTP server on the back end store. However the protocol is used in specific ways. For example:

```
---> LHLO
<--- 250 OK
```

No action is taken on the LHLO message. The reply is always 250 OK.

```
---> MAIL FROM: address size=messageSizeInBytes
<--- 250 OK
```

No checks or conversions are made on the originator address. The size= parameter gives a size in bytes for the message that is to be delivered. This is the size of the message exactly as it appears in the protocol. It is not necessarily the exact size of the message, but the actual message size will not exceed this size. The LMTP server allocates a memory buffer of this size to receive the message.

```
---> RCPT TO: uid+folder@domain xquota=size,number xdflg=xxx
<--- 250 OK
```

No checks are made on the recipient addresses at the time they are received, but a list of recipients is built for later use. Note that the @domain part of the address is omitted for uids in the primary domain, and that the +folder part is optional. This is the same address format used by the message store channel in the MTA.

The xquota= parameter gives the user's message quotas which consist of the maximum total size and the maximum number of messages. The MTA provides this information which it retrieves while performing an LDAP lookup on the user to do the address translation. This information is used to keep the quota information in the message store synchronized with the directory. Getting the quota information does not result in an additional performance hit.

The xdflg= parameter specifies a number which is interpreted as a bit field. These bits control how the message is delivered. For example, the bit whose value is 2, if set, guarantees delivery of the message even if the user is over quota. (Note that xdflg is an internal parameter and the bits in it are subject to change or addition without notice. We do not support other clients using this extension with our server, nor do we support using our client with some other server and this parameter.)

This interaction may be repeated many times, once for each recipient.

```
--->DATA
---> <the message text>
--->.
```

The LMTP client then sends the entire message, dot-stuffed, just as SMTP does. The message finishes with a dot (.) alone on a line. If the message size is exceeded the LMTP server sends:

```
<--- 500 message too big
```

and ends the connection.

Assuming that the message is received correctly, the LMTP server then sends back to the LMTP client the status for each recipient given in the `RCPT TO:` lines. For instance, if the message is delivered successfully, the response is:

```
<--- 250 2.5.0 address OK
```

where `address` is exactly as it appeared on the `RCPT TO:` line.

The conversation can either repeat with another `MAIL FROM:` line or end with the following interaction:

```
---> quit
<--- 221 OK
```

The following table shows the possible status codes for each recipient. This three-column table shows the short code in the first column, its long-code equivalent in the second column and the status text in the third column. 2.x.x status codes are success codes, 4.x.x codes are retryable errors, 5.x.x codes are non retryable errors.

LMTP Status Codes for Recipients

Short Code	Long Code	Status Text
250	2.5.0	OK
420	4.2.0	Mailbox Locked
422	4.2.2	Quota Exceeded
420	4.2.0	Mailbox Bad Formats
420	4.2.0	Mailbox not supported
430	4.3.0	IMAP IOERROR
522	5.2.2	Persistent Quota Exceeded
523	5.2.3	Message too large
511	5.1.1	mailbox nonexistent
560	5.6.0	message contains null
560	5.6.0	message contains nl
560	5.6.0	message has bad header
560	5.6.0	message has no blank line

Otherwise, there are changes to the delivery options for mailbox, native (and, therefore, UNIX), and file. The object of these rules is to generate addresses that will cause the messages to be sent through the appropriate LMTP channel to the back end servers. The addresses generated are source routed addresses of the form:

```
@sourceroute:_localpart_@_domain_
```

Chapter 18. Vacation Automatic Message Reply

Vacation Automatic Message Reply

For automatically generated responses to email (autoreply), specifically vacation messages, the MTA uses Message Disposition Notifications (*MDNs*) and the Sieve scripting language. MDNs are email messages sent by the MTA to a sender and/or postmaster reporting on a message's delivery disposition. MDNs are also known as read receipts, acknowledgments, receipt notifications, or delivery receipts. The Sieve is a simple scripting language used to create mail filters. Starting with Messaging Server 6, the character set used is UTF-8 instead of ISO-2022-JP.

This information describes the vacation autoreply mechanism. In most cases, you do not need to modify the default configuration. However, you might want to configure your system such that an MTA relay performs vacation processing rather than the back-end message stores.

Topics:

Vacation Autoreply Overview

Vacation Sieve scripts are generated automatically from the various LDAP vacation attributes (see Vacation Autoreply Attributes). They can also be specified explicitly for additional flexibility. The underlying mechanism for tracking vacations is a set of files, one per intended recipient, that keep track of when replies were sent to the various senders.

> **Note**
> The charset of vacation message has changed to UTF-8.

By default, the MTA evaluates vacation on the back-end store systems. However, since MTA relays do not do as much work as back-end stores, for performance reasons, you can have the MTA evaluate vacation on the mail relay machines instead of on the back-end store. Use of this feature, however, can result in vacation responses being sent out more often than intended because different relays handle different messages. If you do not want vacation messages to be sent out more often than you intend, you may share the tracking of files between the relays. If this is also unacceptable to you, you can always have vacation evaluated on the back-end store systems.

Configuring Autoreply

Delivery addresses are generated through a set of patterns. The patterns used depend upon the values defined for the `mailDeliveryOption` attribute. A delivery address is generated for each valid `mailDeliveryOption`. The patterns are defined by the MTA option `DELIVERY_OPTIONS`, which are defined in the `option.dat` file. The default autoreply rule in `DELIVERY_OPTIONS` in the `option.dat` file is:

```
*^!autoreply=$M+$D@bitbucket
```

The MTA notes the ^ on the autoreply DELIVERY_OPTION MTA option. This causes the MTA to check the vacation dates. If the current date is within the vacation dates, processing continues and the MTA notes the ! on the autoreply DELIVERY_OPTION. The MTA then creates a vacation Sieve script based on the various autoreply LDAP attributes on the user's entry. The autoreply rule can have the prefix characters !, #, ^, and *.

You could have the ! flag on the mailbox delivery option. This would enable the generation of the vacation script unconditionally. It makes sense, however, to have the autoreply machinery enabled by a separate delivery option so that it can be further gated by the ^ flag. Checking the dates at this stage is more efficient than engaging the Sieve logic.

The following table shows the prefix characters used for the autoreply rule in the first column and their definitions in the second column.

Prefix Characters for the Autoreply Rule in DELIVERY_OPTIONS

Prefix Character	Definition
!	Enable the generation of the autoreply Sieve script.
#	Allows the processing to take place on relays.
^	Option is only evaluated if the vacation dates indicate that it should be evaluated.
@	Extract preferred language information from various message header fields as well as from LDAP entries associated with envelope From: addresses. For this information to be available at the correct time, the message must pass through the reprocess channel when autoreply is engaged. This is done by adding the @ flag to the autoreply delivery option. Note that the addition of a channel hop increases message processing overhead.

The autoreply rule itself specifies an address destined for the bitbucket channel. The mail is considered delivered by this method once the autoreply is generated, but the MTA machinery requires a delivery address. Anything delivered to the bitbucket channel is discarded.

To Configure Autoreply on the Back-end Store System

The default autoreply rule in DELIVERY_OPTIONS causes the autoreply to take place on the mail server that serves the user. If you want vacation messages to be evaluated on the back-end store system, you do not have to configure anything. This is the default behavior.

To Configure Autoreply on a Relay

If you want to evaluate vacation on the relay rather than on the back-end store system to enhance performance, perform the following steps:

1. Edit the option.dat file and prepend the # character to the autoreply rule in DELIVERY_OPTIONS so that it resembles the following:

```
#*^!autoreply=$M+$D@bitbucket
```

Note
Specify the DELIVERY_OPTIONS option in its entirety. There is no facility to override individual rules.

2. After changing the DELIVERY_OPTIONS option, verify that it resembles the following:

```
DELIVERY_OPTIONS=*mailbox=$M%$\$2I$_+$2S@ims-ms-daemon, \
 &members=*, \
 *native=$M@native-daemon, \
 /hold=@hold-daemon:$A, \
 *unix=$M@native-daemon, \
 &file=+$F@native-daemon, \
 &@members_offline=*, \
 program=$M%$P@pipe-daemon, \
 #forward=**, \
 #*^!autoreply=$M+$D@bitbucket
```

Processing now takes place on the relays.

To Share Autoreply Information Between Relays

If the MTA perform autoreplies on the relays, then either each relay can keep track independently of whether a particular correspondent has sent an away message recently, or this information can be shared between the relays. The former case is simpler, especially if having away messages sent out too many times does not matter. If you want strict application of the away message frequency rules, then the information must be shared between relays.

To share the information among the relays, the files must reside on an NFS mounted directory. The location and specific names of these files are determined by the VACATION_TEMPLATE option in the option.dat file. The VACATION_TEMPLATE must be in the form of a file:// URL and specify the path and template for the file names. The default value is:

```
VACATION_TEMPLATE=file:///opt/SUNWmsgsr/data/vacation/$3I/$1U/$2U/$U.vac
```

For an explanation of delivery option rules and meta characters, see Delivery Options Processing.

Use the following steps to share autoreply information among relays:

1. Create and share a directory for these files on an NFS server (for example, /etc/dfs/dfstab).
2. Make sure that the directory on the NFS server is writable by the mailsrv userid.
3. Mount that directory on each relay.

```
# cd /opt/sun/comms/messaging64/data
# mkdir vacfiles
# chown mailsrv:mail vacfiles
# mount -o rw,soft,timeo=2 <nfs-server>:<shared-directory-path> \
     /opt/sun/comms/messaging64/data/vacfiles
```

4. Make sure that the directory is mounted after a reboot (that is, edit the /etc/vfstab file accordingly).
5. Edit the option.dat file as follows.

```
VACATION_TEMPLATE=file:///opt/sun/comms/messaging64/data/vacfiles/$3I/$1U
```

Note
For important information on NFS mount options, see http://msg.wikidoc.info/index.php/Multi-host_defragmentation_channel_operation.

Vacation Autoreply Theory of Operation

The vacation action, when invoked, works as follows:

1. Messaging Server checks to make sure that the vacation action was performed by a user-level rather than a system-level Sieve script. An error results if vacation is used in a system level script.
2. The "no vacation notice" internal MTA flag is checked.
   If it is set, processing terminates and no vacation notice is sent.
3. The return address for the message is now checked.
   If it is blank, processing terminates and no vacation notice is sent.
4. The MTA checks to see if the user's address or any of the additional addresses specified in the `:addresses` tagged argument appear in a `To:`, `Cc:`, `Resent-to:`, or `Resent-cc:` header field for the current message. Processing terminates and no vacation notice is sent if none of the addresses is found in any of the header fields.
5. Messaging Server constructs a hash of the `:subject` argument and the reason string.
   This string, along with the return address of the current message, is checked against a per-user record of previous vacation responses. Processing terminates and no response is sent if a response has already been sent within the time allowed by the `:days` argument.
6. Messaging Server constructs a vacation notice from the `:subject` argument, reason string, and `:mime` argument. Two basic forms for this response message are possible:
   - A message disposition notification of the form specified in RFC 2298, with the first part containing the reason text.
   - A single part text reply. (This form is only used to support the "reply" autoreply mode attribute setting.)

The `mailautoreplymode` is automatically set to `reply` when vacation messages are configured through Messenger Express.

The "no vacation notice" MTA flag is clear by default. It can be set by a system-level Sieve script through the use of the nonstandard `novacation` action. The `novacation` Sieve action is only allowed in a system level Sieve script. It will generate an error if it is used in a user level script. You can use this action to implement site-wide restrictions on vacation replies such as blocking replies to addresses containing the substring "`MAILER-DAEMON`."

Per-user per-response information is stored in a set of flat text files, one per local user. The location and naming scheme for these files is specified by the setting of the `VACATION_TEMPLATE MTA` option. This option should be set to a `file: URL`.

Maintenance of these files is automatic and controlled by the `VACATION_CLEANUP` integer MTA option setting. Each time one of these files is opened, the value of the current time in seconds modulo this value is computed. If the result is zero the file is scanned and all expired entries are removed. The default value for the option is 200, which means that there is 1-in-200 chance that a cleanup pass will be performed.

The machinery used to read and write these flat text files is designed in such a way that it should be able to operate correctly over NFS. This enables multiple MTAs to share a single set of files on a common file system.

Vacation Autoreply Attributes

The set of user LDAP directory attributes that the vacation action uses are:

- Attribute defined by the MTA option LDAP_AUTOREPLY_ADDRESSES
  This attribute provides the ability to generate `:addresses` arguments to sieve vacation. This option has no value by default. The attribute can be multivalued, with each value specifying a separate address to pass to the `:addresses` vacation parameter.
- Attribute defined by LDAP_PERSONAL_NAME
  Alias processing keeps track of personal name information specified in this attribute and will use this information to construct From: fields for any MDNs or vacation replies that are generated. Use with caution to avoid exposing personal information.
- `vacationStartDate`
  Vacation start date and time. The value is in the format YYYYMMDDHHMMSSZ. This value is normalized to GMT. An autoreply should only be generated if the current time is after the time specified by this attribute. No start date is enforced if this attribute is missing. The MTA can be instructed to look at a different attribute for this information by setting the LDAP_START_DATE MTA option to a different attribute name.
  This attribute will be read and checked by the code that generated the Sieve script. Vacation processing will be aborted if the current date is before the vacation start date. This attribute cannot be handled by the script itself because at present Sieve lacks date/time testing and comparison facilities.
- `vacationEndDate`
  Vacation end date and time. The value is in the format YYYYMMDDHHMMSSZ. This value is normalized to GMT. An autoreply should only be generated if the current time is before the time specified by this attribute. No end date is enforced if this attribute is missing. The MTA can be instructed to look at a different attribute for this information by setting the LDAP_END_DATE MTA option to a different attribute name.
  This attribute will be read and checked by the code that generated the Sieve script. Vacation processing will be aborted if the current date is after the vacation end date. This attribute cannot be handled in the script itself because at present Sieve lacks date/time testing and comparison facilities.
- `mailAutoReplyMode`
  Specifies autoreply mode for the user mail account. Valid values of this attribute are:
  - `echo` - Create a multipart that echoes the original message text in addition to the added `mailAutoReplyText` or `mailAutoReplyTextInternal` text.
  - `reply` - Send a single part reply as specified by either `mailAutoReplyText` or `mailAutoReplyTextInternal` to the original sender.
    These modes will appear in the Sieve script as nonstandard `:echo` and `:reply` arguments to the vacation action. `echo` will produce a "processed" message disposition notification (MDN) that contains the original message as returned content. `reply` will produce a pure reply containing only the reply text. An illegal value will not manifest as any argument to the vacation action and this will produce an MDN containing only the headers of the original message. Note also that selecting an autoreply mode of echo causes an automatic reply to be sent to every message regardless of how recently a previous reply was sent.
    The MTA can be instructed to use a different attribute for this information by setting the LDAP_AUTOREPLY_MODE MTA option to a different attribute name.
- `mailAutoReplySubject`
  Specifies the contents of the subject field to use in the autoreply response. This must be a UTF-8 string. This value gets passed as the `:subject` argument to the vacation action. The MTA can be instructed to use a different attribute for this information by setting the LDAP_AUTOREPLY_SUBJECT MTA option to a different attribute name.
- `mailAutoReplyText`
  Autoreply text sent to all senders except users in the recipient's domain. If not specified, external users receive no vacation message. The MTA can be instructed to use a different attribute for this information by setting the LDAP_AUTOREPLY_TEXT MTA option to a different attribute name.
- `mailAutoReplyTextInternal`
  Auto-reply text sent to senders from the recipients domain. If not specified, then internal users get the mail autoreply text message. The MTA can be instructed to use a different attribute for this

information by setting the LDAP_AUTOREPLY_TEXT_INT MTA option to a different attribute name. The MTA passes either the mailAutoReplyText or mailAutoReplyTextInternal attribute value as the reason string to the vacation action.

- mailAutoReplyTimeOut
  Duration, in hours, for successive autoreply responses to any given mail sender. Used only when mailAutoReplyMode=reply. If value is 0 then a response is sent back every time a message is received. This value is converted to the nonstandard :hours argument to the vacation action. (Normally the Sieve vacation action only supports the :days argument for this purpose and does not allow a value of 0.)
  If this attribute doesn't appear on a user entry, a default time-out will be obtained from the AUTOREPLY_TIMEOUT_DEFAULT MTA option. The MTA can be instructed to use a different attribute for this information by setting the LDAP_AUTOREPLY_TIMEOUT MTA option.

The MTA can choose between multiple LDAP attributes and attribute values with different language tags and determine the correct value to use. The language tags in effect are compared against the preferred language information associated with the envelope from address. Currently the only attributes receiving this treatment are LDAP_AUTOREPLY_SUBJECT (normally mailAutoReplySubject), LDAP_AUTOREPLY_TEXT (normally mailAutoReplyText), LDAP_AUTOREPLY_TEXT_INT (normally mailAutoReplyTextInternal), LDAP_SPARE_4, LDAP_SPARE_5, LDAP_PREFIX_TEXT and LDAP_SUFFIX_TEXT.

It is expected that each attribute value will have a different language tag value. If different values have the same tag value the choice between them will be essentially random.

Other Auto Reply Tasks and Issues

This section describes auto reply tasks and issues not described in the configuration section.

To Send Autoreply Messages for Email That Have Been Automatically Forwarded from Another Mail Server

An autoreply problem can occur when the MTA receives a message that has been automatically forwarded from another system in some other administrative domain. For example, if a customer has a home account with sesta.com and the customer sets that account to automatically forward messages to their work account at siroe.com and if siroe.com uses Messaging Server and that user has set his account to autoreply a vacation message, then Messaging Server will have a problem sending out a vacation message.

The problem occurs because the sesta.com mail server changes the envelope To: address from user@sesta.com to user@siroe.com, but it will not change the To: header, which remains user@sesta.com. When the MTA receives the message, it looks at the header address only. It attempts to match this address with an address in the LDAP user directory. If it finds a match with someone who has set autoreply, then a vacation message is sent. Because there is no LDAP address match to user@sesta.com, no vacation message is sent. The problem is that the actual address is in the envelope and not in the header.

Since the recipient's address known to the remote system doing automatic forwarding isn't known to correspond to the user by the local system, there needs to be a way for the recipient to make such addresses known to the local system so vacation replies will be sent when necessary.

The :addresses argument to the Sieve vacation action provides this capability. It accepts a list of addresses that correspond to the recipient for purposes of making this check. The attribute defined by the MTA option LDAP_AUTOREPLY_ADDRESSES allows specification of such addresses in the user's LDAP entry.

To provide autoreply capability for messages that have been automatically forwarded from mail servers in some other administrative domain, the user or administrator would set the email addresses from where

those messages may be forwarded to the attribute defined by LDAP_AUTOREPLY_ADDRESSES.

Chapter 19. Mail Filtering and Access Control

Mail Filtering and Access Control

This information discusses how to filter mail based on its source (sender, IP address and so on) or header strings. Two mail filtering mechanisms are used, controlling access to the MTA by using mapping tables and Sieve server-side rules (SSR).

Limiting access to the MTA by using the mapping tables enables messages to be filtered based on From: and To: addresses, IP address, port numbers, and source or destination channels. Mapping tables permit SMTP relaying to be enabled or disabled. Sieve is a mail filtering script that enables messages to be filtered based on strings found in headers.

If envelope-level controls are desired, use mapping tables to filter mail. If header-based controls are desired, use Sieve server-side rules.

Topics:

PART 1. MAPPING TABLES
Enables the administrator to control access to MTA services by configuring certain mapping tables. The administrator can control who can or cannot send mail, receive mail through Messaging Server.

PART 2. MAILBOX FILTERS
Enables users and administrators to filter messages and specify actions on those filtered messages based on a string found in the message header. Uses the Sieve filter language and can filter at the channel, MTA or user level.

PART 1. MAPPING TABLES

The following topics contain the specifics of each mapping table. See Mappings File for the details about the format and operation, patterns, templates, and metacharacters which are common to all mapping tables.

- Controlling Access with Mapping Tables
- Access Control Mapping Table Flags
- Send Access and Mail Access Mapping Tables
- FROM_ACCESS Mapping Table
- PORT_ACCESS Mapping Table
- IP_ACCESS Mapping Table
- When Access Controls Are Applied
- To Test Access Control Mappings
- To Limit Specified IP Address Connections to the MTA
- To Add SMTP Relaying
- Configuring SMTP Relay Blocking
- Handling Large Numbers of Access Entries
- Controlling the Envelope From: Address in Mappings Strings and Mailing List Named Parameters and LDAP Attributes

Controlling Access with Mapping Tables

You can control access to your mail services by configuring certain mapping tables. These mapping tables allow you to control who can or cannot send mail, receive mail, or both. Access Control Mapping

Tables lists the mapping tables described in this section. The application information string supplied to the FROM_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS mappings includes the system name claimed in the HELO/EHLO SMTP command. This name appears at the end of the string and is separated from the rest of the string (normally "SMTP*") by a slash. The claimed system name can be useful in blocking some worms and viruses.

Access Control Mapping Tables - Operation

Access control mapping tables follow the same general format and operations as all mappings tables. (See Mappings File for more details about the format and operation, and patterns, templates, and metacharacters which apply to all mapping tables.) They consist of a mapping table name, followed by a blank line, followed by one or more mapping entries. Mapping entries consist of a *search pattern* on the left side and a *template* on the right side. The search pattern filters specific messages and the template specifies actions to take on the message. For example:

```
SEND_ACCESS

 *|Elvis1@sesta.com|*|*      $Y
 *|Nelson7@sesta.com|*|*     $Y
 *|AkiraK@sesta.com|*|*      $Y
 *|*@sesta.com|*|*           $NMail$ Blocked
```

In this example all email from the domain sesta.com except those of Elvis1, Nelson, AkiraK are blocked.

The search pattern for access control mapping entries consist of a number of search criteria separated by vertical bars (|). The order of the search criteria depends on the access mapping table and is described in subsequent sections. But as an example, the SEND_ACCESS mapping table has the following search form:

```
src-channel|from-address|dst-channel|to-address
```

where *src-channel* is the channel queueing the message; *from-address* is the address of the message's originator; *dst-channel* is the channel to which the message will be queued; and *to-address* is the address to which the message is addressed. Use of an asterisk in any of these four fields causes that field to match any channel or address, as appropriate.

The *from-address* is a positional content string placeholder. In some circumstances, you can modify the default type of information, for example, by using the use_auth_return MTA option (introduced in Messaging Server 7.0). An option might affect only substrings in certain mappings and might be subject to precedence rules of options affecting the same mapping table substring. For more information, see Controlling the Envelope From: Address in Mappings Strings and Mailing List Named Parameters and LDAP Attributes.

Note
Whenever the mappings file is modified, you must recompile the configuration. See Compiling the MTA Configuration.

Mapping Table	Description
SEND_ACCESS	Used to block incoming connections based on envelope From address, envelope To address, source and destination channels. The To address is checked after rewriting, alias expansion, and so on, have been performed.
ORIG_SEND_ACCESS	Used to block incoming connections based on envelope From address, envelope To address, source and destination channels. The *original* To address after rewriting but before alias expansion is used, but the function of alias expansion has already been performed.
MAIL_ACCESS	Used to block incoming connections based on combined information found in SEND_ACCESS and PORT_ACCESS tables: that is, the channel and address information found in SEND_ACCESS combined with the IP address and port number information found in PORT_ACCESS.
ORIG_MAIL_ACCESS	Used to block incoming connections based on combined information found in ORIG_SEND_ACCESS and PORT_ACCESS tables: that is, the channel and address information found in ORIG_SEND_ACCESS combined with the IP address and port number information found in PORT_ACCESS.
FROM_ACCESS	Used to filter mail based on envelope From addresses. Use this table if the To address is irrelevant.
PORT_ACCESS	Used to block incoming connections based on IP address and TCP port.
IP_ACCESS.	Used to control outgoing connections based on IP address information.

The MAIL_ACCESS and ORIG_MAIL_ACCESS mappings are the most general, having available not only the address and channel information available to SEND_ACCESS and ORIG_SEND_ACCESS, but also any information that would be available via the PORT_ACCESS mapping table, including IP address and port number information.

Access Control Mapping Table Flags

This section consists of the following subsections:

Send Access and Mail Access Mapping Tables shows the access mapping flags and metacharacters relevant for the SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS. Also see FROM_ACCESS Mapping Table, PORT_ACCESS Mapping Table, and IP_ACCESS Mapping Table .

See Mappings File for more details about the format and operation, and patterns, templates, and metacharacters which apply to all mapping tables.

Input vs Output Flags

Mapping table templates can test input flags and can set output flags. Input and output flags should be thought of as being stored in separate namespaces. For example, the $A input flag may have nothing to do with a $A output flag. "$A" is a short hand way of saying "the A flag" (as opposed to the string "A").

There are input flags set by the MTA when it does a probe – so there is the probe string, plus any probe input flags. Then there is the output string plus output flags that were set on the right hand side of the mapping entry.

So $A could be set by the MTA as an input flag, then testable (on the right hand side of an entry) via `$:A` or `$;A`. Then the right hand side of an entry may set its own output flags (that is, $A) which as output means something entirely different – and which do not count as input flags for testing purposes (regardless of whether the mapping "iterates" – say with $C). That is, the flag tests such as `$:`*flag-char* are always on the *original* (set by the MTA, or via `-flags` input setting in `imsimta test -mapping`) input flags, not on any flags that may have been set in the right-hand-side of the mapping.

For example, try using `imsimta test -mapping`, especially with its `-flags` switch to set some input flags. Try using some arbitrary, private mapping table. Use a few `$:`*flag* or `$;`*flag* tests on your right hand side. Also notice that the output lists (separately):

```
Output string: ...
Output flags: [...]
```

where the output flags (the stuff inside the []'s) correspond to the output flags set. That is, with a mapping:

```
X-FLAGS

   *              $C$:AA$ set$Y
   *              $;AA$ not$ set$Y
```

try something like:

```
# imsimta test -mapping -flags=A -table=x-flags
```

and note how it yields

```
Output string: A set
Output flags: [0, 'Y' (89)]
```

meaning that it output the string "A set", matched on the 0th iteration (as opposed to iterative or recursive mappings where you'll see the number go up) and output the Y flag (or $Y if you want to write it that way) but the A flag that was an input flag is *not* an output flag. 'Y' is character position 89 in ASCII – the character position is shown for extra clarity.

In contrast,

```
 # imsimta test -mapping -flags=B -table=x-flags
```

yields

```
Output string: A not set
Output flags: [0, 'Y' (89)]
```

Output Flag Argument Order

Flags with arguments must have those arguments arranged in the reading order shown in the table. For example:

```
ORIG_SEND_ACCESS

  tcp_local|*|tcp_local|*     $N$D30|Relaying$ not$ allowed
```

In this case, the proper order is the delay period followed by the rejection string. Note that the flags themselves can be in any order. So the following entries have identical results:

```
30|Relaying$ not$ allowed$D$N
$N30|Relaying$ not$ allowed$D
30|$N$DRelaying$ not$ allowed
```

For the details of the flags and metacharacters supported in each table, see:

- Send Access and Mail Access Mapping Tables
- FROM_ACCESS Mapping Table
- PORT_ACCESS Mapping Table
- IP_ACCESS Mapping Table

Send Access and Mail Access Mapping Tables

Access Mapping Flags and Metacharacters

The following flags and metacharacters apply to the `SEND_ACCESS`, `ORIG_SEND_ACCESS`, `MAIL_ACCESS`, and `ORIG_MAIL_ACCESS` mapping tables. The `FROM_ACCESS` and `PORT_ACCESS` mapping tables have different sets of flags – see the FROM_ACCESS Mapping Table and PORT_ACCESS Mapping Table sections.

- Send and Mail Access Input Flags
- Send and Mail Access Output Flags
- SEND_ACCESS and ORIG_SEND_ACCESS Mapping Tables
- MAIL_ACCESS and ORIG_MAIL_ACCESS Mapping Tables

Send and Mail Access Input Flags

Input flags are set by the process that calls the mapping table and tested for by using the `$:` and `$;` metacharacters in the template. The `$:` will succeed if the flag is set or fail if it is not set. The `$;` will succeed if the flag is not set or fail if it is set. The following table shows the `$:`*x* form to test if the flag is set. The `$;`*x* form can also be used to test if condition is not true.

Flag	Description
$:|	Match only if external material (that is, an envelope address) in the probe contained a vertical bar. This feature was introduced in **Messaging Server 7.**
$:A	Match only if SASL (SMTP AUTH) has been used
$:D	Match only if DELAY delivery receipts have been requested (that is, NOTIFY=DELAY)
$:E	Match only if ESMTP has been used. This feature was introduced in **Messaging Server 6.3.**
$:F	Match only if FAILURE delivery receipts have been requested (that is, NOTIFY=FAILURE)
$:L	Match only if LMTP has been used. This feature was introduced in **Messaging Server 6.3.**
$:P	Match only if POP-before-SMTP was used. This feature was introduced in **Messaging Server 7.**
$:S	Match only if SUCCESS delivery receipts have been requested (that is, NOTIFY=SUCCESS)
$:T	Match only if TLS has been used
$:V	Match only if recipient address expanded via an alias. This feature was introduced in **Messaging Server 7 Update 1.**

Send and Mail Access Output Flags

Output flags are set by the template in the mapping table entry and consumed upon return to the calling program.

Flag	Description
$.	Establishes a string which will be processed as the mapping entry result in the event of a temporary LDAP lookup failure. By default a temporary failure string remains set only for the duration of the current rule. "$.." can be used to return to the default state where no temporary failure string is set and temporary LDAP failures cause mapping entry or rewrite rule failure. Note that all errors other than failure to match an entry in the directory are considered to be temporary errors; in general it isn't possible to distinguish between errors caused by incorrect LDAP URLs and errors caused by directory server configuration problems. This feature was introduced in **Messaging Server 6.3.**
$*	If used with $N, force disconnect of the SMTP session. This feature was introduced in **Messaging Server 7 Update 2.**
$B	Redirect the message to the bitbucket; the effect is per-recipient in the Send/Mail Access mapping tables, vs entire message in the FROM_ACCESS Mapping Table.
$H	Hold the message as a .HELD file; the effect is per-recipient in the Send/Mail Access mapping tables, vs entire message in the FROM_ACCESS Mapping Table.
$J	Used with $M (see below) to cause generation of envelope "journal" format rather than the default DSN encapsulated format for the "capture" message copy (added in 7u3p16)
$O	Forces single-copy-per-recipient message copy "split up", as if the `single` channel keyword were set on the relevant destination channel(s).
$P	Force to enqueue to the reprocess channel. (For instance, this might be useful as part of `$. text.` handling when attempting an LDAP lookup that encountered an LDAP directory temporary problem; that is, in case of an LDAP lookup problem, defer to the reprocess channel.)
$V	Force Sieve filter discard behavior for all recipients of this message copy.
$v	Force Sieve filter discard behavior for solely this recipient. This feature was introduced in **Messaging Server 7 Update 3.**
$Y	Allow access.
$Z	Force Sieve filter jettison behavior (non-overridable discard) for all recipients of this message copy.
$z	Force Sieve filter jettison behavior (non-overridable discard) for solely this recipient. This feature was introduced in **Messaging Server 7 Update 3.**

Flags with Arguments, in Output Flag Argument Order+ (DO NOT ALPHABETIZE THIS LIST!)

Flag	Description
$U*integer*	Enable channel (slave) debugging; if the optional n argument is specified, it also sets the specified value for enqueue debugging (see MM_DEBUG option).
$I*user*| *identifier*	Check specified user for specified groupid (UNIX) and if not in the group, reject access (effectively sets the $N output flag).
$<*string*	+++ Send *string* to UNIX syslog (`user.notice` facility and severity) if probe matches. (see SNDOPR_PRIORITY option)
$>*string*	+++ Send *string* to UNIX syslog (`user.notice` facility and severity) if access is rejected. (see SNDOPR_PRIORITY option)

$D*delay*	Delay response for an interval of *delay* hundredths of seconds; a positive value causes the delay to be imposed on each command in the transaction; a negative value causes the delay to be imposed only on the address handover (SMTP MAIL FROM: command for the FROM_ACCESS table; SMTP RCPT TO: command for the other tables).
$T*tag*	Prefix subsequent **ACCESS mapping table probes with tag _tag*
$A*header*	Add the header line *header* to the message.
$G *conversion_tag*	If used in ORIG_SEND_ACCESS, SEND_ACCESS, ORIG_MAIL_ACCESS, and MAIL_ACCESS, it reads a value from the mapping result and treats it as a set of conversion tags to be applied to the current recipient. If used with FROM_ACCESS, conversion tags are applied to all recipients. $G is positioned after $A (header address) in the sequence of arguments read from the mappings. See Mail Conversion Tags. This feature was introduced in **Messaging Server 6.0.**
$M*address*	Capture a copy of the message, sending the captured copy to *address*. By default, this captured copy is DSN encapsulated---but see the no-argument, non-FROM_ACCESS $J output flag above.
$, *spamadjust_arg*	Set a spam level value x (between -10000.0 and 10000.0). If the message already had a spam level, this is a spamadjust effect (adds or subtracts the specified amount x from the prior spam level). Note that such a spam level/spamadjust effect applies to all recipients (even for the recipient-specific mapping tables such as SEND_ACCESS) in order for tests to see if one of the recipients is a "honeypot" address. Allows you to perform a sieve spamadjust operation from the access mapping tables. Argument takes the same form as a spamadjust argument. Note also that some of these mappings are applied on a per-recipient basis. Any spamadjust operation that is done applies to all recipients. This feature was introduced in **Messaging Server 6.1.**
$+E*name*|*value*	Set the environment variable *name* to *value*. This feature was introduced in **Messaging Server 7 Update 2.**
$X*error-code*	Lets you specify the extended SMTP *error-code* (the digit.digit.digit part), and if the first digit is a 4 rather than a 5, then you'll get a 452 SMTP temporary error, rather than the usual 550 SMTP permanent error. For example: ORIG_SEND_ACCESS <...probe...> NX4.5.9|Temporary$ problem$ with$ address;$ try$ later$
$N*string* $F*string*	Reject access with the optional error text *string*. $F and $N are synonyms.
$S*url*	Apply the Sieve filter obtained from resolving *url*.
$+R*n*|*string*	Opt in to spam filtering. Only applied if neither $N nor $F is set. *n* is which spam/virus filter package; *string* is the optin string to pass to that spam filter. This feature was introduced in **Messaging Server 7 Update 2.**

{+} To use multiple flags with arguments, separate the arguments with the vertical bar character, |, placing the arguments in the order listed in this table.
+++ It is a good idea to use the $D flag when dealing with problem senders, to prevent a denial of service attack. In particular, it is a good idea to use $D in any $> entry or < entry rejecting access.

SEND_ACCESS and ORIG_SEND_ACCESS Mapping Tables

You can use the SEND_ACCESS and ORIG_SEND_ACCESS mapping tables to control who can or cannot send mail, receive mail, or both. The access checks have available a message's envelope From: address and envelope To: addresses, and knowledge of what channel the message came in, and what channel it would attempt to go out.

If a SEND_ACCESS or ORIG_SEND_ACCESS mapping table exists, then for each recipient of every message passing through the MTA, the MTA will scan the table with a string of the following form (note the use of the vertical bar character, |):

```
src-channel|from-address|dst-channel|to-address
```

The *src-channel* is the channel queueing the message; *from-address* is the address of the message's originator; *dst-channel* is the channel to which the message will be queued; and *to-address* is the address to which the message is addressed. Use of an asterisk in any of these four fields causes that field to match any channel or address, as appropriate.

The addresses here are envelope addresses; that is, envelope From: address and envelope To: address. In the case of SEND_ACCESS, the envelope To: address is checked after rewriting, alias expansion, etc., have been performed; in the case of ORIG_SEND_ACCESS the originally specified envelope To: address is checked after rewriting, but before alias expansion. Note however that the function of alias expansion has already been performed at this point, the ORIG_* variant simply uses a copy of the address from before expansion.

If the search string matches a pattern (that is, the left-hand side of an entry in the table), then the resulting output of the mapping is checked. If the output contains the flags $Y or $y, then the enqueue for that particular To: address is permitted. If the output contains any of the flags $N, $n, $F, or $f, then the enqueue to that particular address is rejected. In the case of a rejection, optional rejection text may be supplied in the mapping output. This string will be included in the rejection error the MTA issues. If no string is output (other than the $N, $n, $F, or $f flag), then default rejection text will be used. For descriptions of additional flags, see Access Control Mapping Table Flags.

See Send Access and Mail Access Mapping Tables for the complete list of flags which apply to the the SEND_ACCESS and ORIG_SEND_ACCESS mapping tables.

Setting the MTA option {ACCESS_ORCPT to 1 adds an additional vertical bar delimited field to the probe value passed to the SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS mapping tables that contains the original recipient (ORCPT) address. If the message doesn't have an ORCPT address, the original unmodified RCPT TO: address is used instead. The default is 0, and the probe value is at the end:

```
src-channel|from-address|dst-channel|to-address|ORCPT_address
```

In the following example, mail sent from UNIX user agents such as mail, Pine, and so on, originates from the local, l, channel and messages to the Internet go out a TCP/IP channel of some sort. Suppose that local users, with the exception of the postmaster, are not allowed to send mail to the Internet but can receive mail from there. Then the SEND_ACCESS mapping table shown in the example below is one possible way to enforce this restriction. In the mapping table, the local host name is assumed to be sesta.com. In the channel name "tcp_*," a wild card is used so as to match any possible TCP/IP channel name (for example, tcp_local).

Example - SEND_ACCESS Mapping Table

```
SEND_ACCESS

   *|postmaster@sesta.com|*|*    $Y
   *|*|*|postmaster@sesta.com    $Y
   l|*@sesta.com|tcp_*|*         $NInternet$ postings$ are$ not$
permitted
```

In the rejection message, dollar signs are used to quote spaces in the message. Without those dollar signs, the rejection would be ended prematurely and only read "Internet" instead of "Internet postings are not permitted". Note that this example ignores other possible sources of "local" postings such as from PC-based mail systems or from POP or IMAP clients.

Note -
The client attempting to send the message determines whether the MTA rejection error text is actually presented to the user who attempted to send the message. If SEND_ACCESS is used to reject an incoming SMTP message, the MTA merely issues an SMTP rejection code including the optional rejection text; it is up to the sending SMTP client to use that information to construct a bounce message to send back to the original sender.

MAIL_ACCESS and ORIG_MAIL_ACCESS Mapping Tables

The MAIL_ACCESS mapping table is a superset of the SEND_ACCESS and PORT_ACCESS mapping tables. It combines both the channel and address information of SEND_ACCESS with the IP address and port number information of PORT_ACCESS. Similarly, the ORIG_MAIL_ACCESS mapping table is a superset of the ORIG_SEND_ACCESS and PORT_ACCESS mapping tables. The format for the probe string for MAIL_ACCESS is:

```
port-access-probe-info|app-info|submit-type|send_access-probe-info
```

Similarly, the format for the probe string for ORIG_MAIL_ACCESS is:

```
port-access-probe-info|app-info|submit-type|orig_send_access-probe-info
```

Here *port-access-probe-info* consists of all the information usually included in a PORT_ACCESS mapping table probe in the case of incoming SMTP messages; otherwise, it is blank. *app-info* includes the system name claimed in the HELO/EHLO SMTP command. This name appears at the end of the string and is separated from the rest of the string (normally "SMTP*") by a slash. The claimed system name can be useful in blocking some worms and viruses. *submit-type* may be one of MAIL, SEND, SAML, or SOML, corresponding to how the message was submitted into Messaging Server. Normally the value is MAIL, meaning it was submitted as a message; SEND, SAML, or SOML can occur in the case of broadcast requests (or combined broadcast/message requests) submitted to the SMTP server. And for the MAIL_ACCESS mapping, *send-access-probe-info* consists of all the information usually included in a SEND_ACCESS mapping table probe. Similarly for the ORIG_MAIL_ACCESS mapping, *orig-send-access-probe-info* consists of all the information usually included in an ORIG_SEND_ACCESS mapping table probe.

See Send Access and Mail Access Mapping Tables for the list of flags which apply to the the MAIL_ACCESS and ORIG_MAIL_ACCESS mapping tables.

See Mappings File for more details about the format and operation, and patterns, templates, and metacharacters which apply to all mapping tables.

Setting the MTA option ACCESS_ORCPT to 1 adds an additional vertical bar delimited field to the probe value passed to the SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS mapping tables that contains the original recipient (ORCPT) address. If the message doesn't have an ORCPT address, the original unmodified RCPT TO: address is used instead. The default is 0, and the probe value is at the end. Example:

```
port-access-probe-info|app-info|submit-type|send_access-probe-info|ORCPT_addres
```

Having the incoming TCP/IP connection information available in the same mapping table as the channel and address information makes it more convenient to impose certain sorts of controls, such as enforcing what envelope From: addresses are allowed to appear in messages from particular IP addresses. This can be desirable to limit cases of email forgery, or to encourage users to configure their POP and IMAP clients' From: address appropriately. For example, a site that wishes to allow the envelope From: address vip@siroe.com to appear only on messages coming from the IP address 1.2.3.1 and 1.2.3.2, and to ensure that the envelope From: addresses on messages from any systems in the 1.2.0.0 subnet are from siroe.com, might use a MAIL_ACCESS mapping table as shown in the following example.

Example - MAIL_ACCESS Mapping Table

```
MAIL_ACCESS

! Entries for vip's two systems
!
  TCP|*|25|1.2.3.1|*|SMTP*|MAIL|tcp_*|vip@siroe.com|*|*  $Y
  TCP|*|25|1.2.3.2|*|SMTP*|MAIL|tcp_*|vip@siroe.com|*|*  $Y
!
! Disallow attempts to use vip's From: address from other
! systems
!
  TCP|*|25|*|*|SMTP*|MAIL|tcp_*|vip@siroe.com|*|*  \
      $N500$ Not$ authorized$ to$ use$ this$ From:$ address
!
! Allow sending from within our subnet with siroe.com From:
! addresses
!
  TCP|*|25|1.2.*.*|*|SMTP*|MAIL|tcp_*|*@siroe.com|*|*  $Y
!
! Allow notifications through
!
  TCP|*|25|1.2.*.*|*|SMTP*|MAIL|tcp_*||*|*  $Y
!
! Block sending from within our subnet with non-siroe.com
! addresses
!
  TCP|*|25|1.2.*.*|*|SMTP*|MAIL|tcp_*|*|*|*  \
     $NOnly$ siroe.com$ From:$ addresses$ authorized
```

FROM_ACCESS Mapping Table

The following flags and metacharacters apply to the FROM_ACCESS mapping table. The SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS mapping tables have a similar set of flags but there are significant differences. The PORT_ACCESS Mapping Table has a different set of flags.

- FROM_ACCESS Description
- FROM_ACCESS Input Flags
- FROM_ACCESS Output Flags

See Mappings File for more details about the format and operation, and patterns, templates, and metacharacters which apply to all mapping tables.

FROM_ACCESS Mapping Table Flags and Metacharacters

FROM_ACCESS Input Flags

Input flags are set by the process that calls the mapping table and tested for by using the `$:` and `$;` metacharacters in the template. The `$:` will succeed if the flag is set or fail if it is not set. The `$;` will succeed if the flag is not set or fail if it is set. The following table shows the `$:`*x* form to test if the flag is set. The `$;`*x* form can also be used to test if condition is not true.

Flag	Description
`$:|`	Match only if external material (that is, an envelope address) in the probe contained a vertical bar. This feature was introduced in **Messaging Server 7.**
`$:A`	Match only if SASL (SMTP AUTH) has been used
`$:E`	Match only if ESMTP has been used. This feature was introduced in **Messaging Server 6.3**.
`$:L`	Match only if LMTP has been used. This feature was introduced in **Messaging Server 6.3.**
`$:P`	Match only if POP-before-SMTP was used. This feature was introduced in **Messaging Server 7.**
`$:T`	Match only if TLS has been used

FROM_ACCESS Output Flags

Output flags are set by the template in the mapping table entry and consumed upon return to the calling program.

Flag	Description
$.	Establishes a string which will be processed as the mapping entry result in the event of a temporary LDAP lookup failure. By default a temporary failure string remains set only for the duration of the current rule. "$.." can be used to return to the default state where no temporary failure string is set and temporary LDAP failures cause mapping entry or rewrite rule failure. Note that all errors other than failure to match an entry in the directory are considered to be temporary errors; in general it isn't possible to distinguish between errors caused by incorrect LDAP URLs and errors caused by directory server configuration problems. This feature was introduced in **Messaging Server 6.3.**
$/	Set the "fast disconnect" flag for sessions that have not yet succeeded in starting a transaction; for such sessions, any subsequent disconnect is done with SO_LINGER enabled and a timeout of 0, which may clear slots quicker on intermediate firewalls and proxies. This feature was introduced in **Messaging Server 7.**
$!	Disable (Sieve requested) vacation messages regarding this message
$*	If used with $N, force disconnect of the SMTP session. This feature was introduced in **Messaging Server 7** for FROM_ACCESS.
$B	Redirect the message to the bitbucket; effects the entire message (all recipients) when used in FROM_ACCESS, vs. per-recipient in the recipient address based *_ACCESS mapping tables.
$H	Hold the message as a .HELD file; effects the entire message (all recipients) when used in FROM_ACCESS, vs. per-recipient in the recipient address based *_ACCESS mapping tables.
$0	Forces single-copy-per-recipient message copy "split up", as if the `single` channel keyword were set on the relevant destination channel(s).
$P	Force to enqueue to the reprocess channel. (For instance, this might be useful as part of `$. text.` handling when attempting an LDAP lookup that encountered an LDAP directory temporary problem; that is, in case of an LDAP lookup problem, defer to the reprocess channel.)
$V	Force Sieve filter discard behavior for all recipients of this message copy.
$Y	Allow access.
$Z	Force Sieve filter jettison behavior (non-overridable discard) for all recipients of this message copy.
$z	Force Sieve filter jettison - synonymous with $Z Also see per-recipient behavior in the recipient address based *_ACCESS mapping tables.

Flags with Arguments, in Output Flag Argument Order+ (DO NOT ALPHABETIZE THIS LIST!)

Flag	Description
$U*integer*	Enable channel (slave) debugging; if the optional n argument is specified, it also sets the specified value for enqueue debugging (see MM_DEBUG option).
$~*channel-name*	Change source channel to _channel-nam_e. This may be of especial interest for the case of incoming notification messages (incoming messages with empty envelope From); note that when this feature is used and it actually changes the source channel, then the FROM_ACCESS check process is restarted; also, $~ can only be applied once. This feature was introduced in **Messaging Server 7.0 and 6.3p1.**
$J*address*	Replace original envelope `From:` address with specified *address*.

$K*address*	Replace the MTA's internal sender address with specified *address* for subsequent checking. Note: to also add a Sender: header, the source channel must include the `authrewrite` keyword.
$I*user|identifier*	Check specified user for specified groupid (UNIX) and if not in the group, reject access (effectively sets the $N output flag).
$<*string*	+++ Send *string* to UNIX syslog (`user.notice` facility and severity) if probe matches. (see SNDOPR_PRIORITY option)
$>*string*	+++ Send *string* to UNIX syslog (`user.notice` facility and severity) if access is rejected. (see SNDOPR_PRIORITY option)
$D*delay*	Delay response for an interval of *delay* hundredths of seconds; a positive value causes the delay to be imposed on each command in the transaction; a negative value causes the delay to be imposed only on the address handover (SMTP MAIL FROM: command for the `FROM_ACCESS` table; SMTP RCPT TO: command for the other tables).
$T*tag*	Prefix subsequent **ACCESS mapping table probes with tag _tag*
$A*header*	Add the header line *header* to the message.
$G*conversion_tag*	If used in `ORIG_SEND_ACCESS`, `SEND_ACCESS`, `ORIG_MAIL_ACCESS`, and `MAIL_ACCESS`, it reads a value from the mapping result and treats it as a set of conversion tags to be applied to the current recipient. If used with `FROM_ACCESS`, conversion tags are applied to all recipients. `$G` is positioned after `$A` (header address) in the sequence of arguments read from the mappings. See Mail Conversion Tags. This feature was introduced in **Messaging Server 6.0.**
$M*address*	Capture a copy of the message, sending the captured copy to *address*. By default, this captured copy is DSN encapsulated---but see the no-argument, non-FROM_ACCESS $J flag above.
$S*x,y,z*	Set a blocklimit, and optionally recipientlimit, and optionally recipientcutoff for the transaction. Prior to MS 6.3, these values are minimized with whatever other such limits may already be in effect; as of MS 6.3 these values override any global MTA option or source-based such limits that may already be effect (but destination-based limits will still be applied, later). See the topic on specifying absolute message size limits in Configuring Channel Definitions.
$`,`*spamadjust_arg*	Set a spam level value x (between -10000.0 and 10000.0). If the message already had a spam level, this is a spamadjust effect (adds or subtracts the specified amount x from the prior spam level). Note that such a spam level/spamadjust effect applies to all recipients (even for the recipient-specific mapping tables such as SEND_ACCESS) in order for tests to see if one of the recipients is a "honeypot" address. Allows you to perform a sieve `spamadjust` operation from the access mapping tables. Argument takes the same form as a spamadjust argument. Note also that some of these mappings are applied on a per-recipient basis. Any spamadjust operation that is done applies to all recipients. This feature was introduced in **Messaging Server 6.1.**
$(*postmaster-address*	Set an override postmaster address. This feature was introduced in **Messaging Server 6.3.**
$) *postmaster-address*	Set an override postmaster address if none was previously set. This feature was introduced in **Messaging Server 6.3.**
$+E*name|value*	Set the environment variable *name* to *value*. This feature was introduced in **Messaging Server 7 Update 2.**

$+R*n*|*string*	Opt in to spam filtering. *n* specifies which spam/virus filter package; *string* is the optin string to pass to that spam filter.

$X*error-code*	Lets you specify the extended SMTP *error-code* (the digit.digit.digit part), and if the first digit is a 4 rather than a 5, then you'll get a 452 SMTP temporary error, rather than the usual 550 SMTP permanent error. For example:
	`ORIG_SEND_ACCESS`
	
	`<...probe...> $N$X4.5.9|Temporary$ problem$ with$ address;$ try$ later$`

$N*string* $F*string*	Reject access with the optional error text *string*. `$F` and `$N` are synonyms.

{+} To use multiple flags with arguments, separate the arguments with the vertical bar character, `|`, placing the arguments in the order listed in this table.
+++ It is a good idea to use the `$D` flag when dealing with problem senders, to prevent a denial of service attack. In particular, it is a good idea to use `$D` in any $> entry or < entry rejecting access.

FROM_ACCESS Description

The `FROM_ACCESS` mapping table may be used to control who can send mail, or to override purported `From:` addresses with authenticated addresses, or both.

The input probe string to the `FROM_ACCESS` mapping table is similar to that for a `MAIL_ACCESS` mapping table, minus the destination channel and address, and with the addition of authenticated sender information, if available. Thus, if a `FROM_ACCESS` mapping table exists, then for each attempted message submission, Messaging Server will search the table with a string of the form (note the use of the vertical bar character, `|`):

```
port-access-probe-info|app-info|submit-type|src-channel|from-address|auth-f
```

Here *port-access-probe-info* consists of all the information usually included in a `PORT_ACCESS` mapping table probe in the case of incoming SMTP messages; otherwise, it is blank. *app-info* includes the system name claimed in the `HELO`/`EHLO` SMTP command. This name appears at the end of the string and is separated from the rest of the string (normally "SMTP*") by a slash. The claimed system name can be useful in blocking some worms and viruses. *submit-type* may be one of MAIL, SEND, SAML, or SOML, corresponding to how the message was submitted into the MTA. Normally the value is MAIL, meaning it was submitted as a message; SEND, SAML, or SOML can occur in the case of broadcast requests (or combined broadcast/message requests) submitted to the SMTP server. *src-channel* is the channel originating the message (that is, queuing the message); *from-address* is the address of the message's purported originator; and *auth-from* is the authenticated originator address, if such information is available, or blank if no authenticated information is available.

If the probe string matches a pattern (that is, the left-hand side of an entry in the table), the resulting output of the mapping is checked. If the output contains the flags $Y or $y, then the enqueue for that particular `To:` address is permitted. If the output contains any of the flags $N, $n, $F, or $f, then the enqueue to that particular address is rejected. In the case of a rejection, optional rejection text may be supplied in the mapping output. This string will be included in the rejection error Messaging Server issues. If no string is output (other than the $N, $n, $F, or $f flag), then default rejection text will be used. For descriptions of additional flags, see Access Control Mapping Table Flags.

Besides determining whether to allow a message to be submitted based on the originator, `FROM_ACCESS` can also be used to alter the envelope `From:` address via the `$J` flag, or to modify the effect of the `authrewrite` channel keyword (adding a `Sender:` header address on an accepted message) via the `$K` flag. For instance, this mapping table can be used to cause the original envelope `From:` address to simply be replaced by the authenticated address.

Example - FROM_ACCESS Mapping Table

```
FROM_ACCESS

  *|SMTP*|*|tcp_auth|*|       $Y
  *|SMTP*|*|tcp_auth|*|*      $Y$J$4
```

When using the FROM_ACCESS mapping table to modify the effect on having `authrewrite` set to a nonzero value on some source channel, it is not necessary to use FROM_ACCESS if the authenticated address is going to be used verbatim.

For example, with `authrewrite 2` set on the `tcp_local` channel, the following FROM_ACCESS mapping table would not be necessary because `authrewrite` alone is sufficient to get this effect (adding the authenticated address verbatim):

```
FROM_ACCESS

   *|SMTP*|*|tcp_auth|*|     $Y
   *|SMTP*|*|tcp_auth|*|*    $Y$K$4
```

However, the real purpose of FROM_ACCESS is to permit more complex and subtle alterations, as shown in the example below. The `authrewrite` keyword alone is appropriate if you want to add a `Sender:` header line (showing the SMTP AUTH authenticated submitter address) to incoming messages. However, suppose you want to force the addition of such a `Sender:` header line to incoming messages only if the SMTP AUTH authenticated submitter address differs from the envelope `From:` address (that is, not bother to add a `Sender:` header line if the addresses match), and suppose further that you wish the SMTP AUTH and envelope `From:` addresses will not be considered to differ merely because the envelope `From:` includes optional subaddress information.

```
FROM_ACCESS

! If no authenticated address is available, do nothing
  *|SMTP*|*|tcp_auth|*| $Y
! If authenticated address matches envelope From:, do nothing
  *|SMTP*|*|tcp_auth|*|$3* $Y
! If authenticated address matches envelope From: sans
! subaddress, do nothing
  *|SMTP*|*|tcp_auth|*+*@*|$3*@$5* $Y
! Fall though to...
! ...authenticated address present, but didn?t match, so force
! Sender: header
  *|SMTP*|*|tcp_auth|*|* $Y$K$4
```

The `$(` metacharacter in a FROM_ACCESS specifies that an address should be read from the result string and used to replace the current overriding postmaster address. `$)` has the same effect with the added constraint that the overriding postmaster address must not be set prior to invoking the mapping. This allows for specific postmaster addresses to be used with addresses in nonlocal domains - domain postmaster addresses by definition only work with locally defined domains. The override address is (currently) the last string read from the FROM_ACCESS result prior to reading any $N/$F failure result.

PORT_ACCESS Mapping Table

The Dispatcher is able to selectively accept or reject incoming connections based on IP address and port number. As the Dispatcher accepts incoming connections, it searches the PORT_ACCESS table using probes of the form:

```
TCP|server-address|server-port|client-address|client-port
```

The Dispatcher tries to match against all PORT_ACCESS mapping entries. If the result of the mapping contains $N or $F, the connection will be immediately closed. Any other result of the mapping indicates that the connection is to be accepted. $N or $F may optionally be followed by a rejection message. If present, the message will be sent back down the connection just prior to closure. Note that a CRLF terminator will be appended to the string before it is sent back down the connection.

The tcp_smtp_server and tcp_lmtp_server processes also probe the PORT_ACCESS table. Checks that should not be duplicated in both the dispatcher and SMTP or LMTP server processes should be qualified using $: or $; and the A or S flags. See PORT_ACCESS Input Flags for more information.

> **Note**
> The MMP does not make use of mapping tables. If you wish to reject SMTP connections from certain IP addresses and you are using the MMP, you must use the TCPAccess option. See To Configure Mail Access with MMP.

To control SMTP connections using mapping tables, use the INTERNAL_IP mapping table (see Allowing SMTP Relaying for External Sites.

The flag $< followed by an optional string causes Messaging Server to send the string to syslog (UNIX) or to the event log (NT) if the mapping probe matches. The flag $> followed by an optional string causes Messaging Server to send the string as to syslog (UNIX) or to the event log (NT) if access is rejected. If bit 1 of the LOG_CONNECTION MTA option is set and the $N flag is set so that the connection is rejected, then also specifying the $T flag will cause a "T" entry to be written to the connection log. If bit 4 of the LOG_CONNECTION MTA option is set, then site-supplied text may be provided in the PORT_ACCESS entry to include in the "C" connection log entries. To specify such text, include two vertical bar characters in the right-hand side of the entry, followed by the desired text. PORT_ACCESS Mapping Flags lists the available flags.

In earlier versions of Messaging Server (6.2 and before) the PORT_ACCESS mapping was only reevaluated by the SMTP server (as opposed to the dispatcher) when bit 4 (value 16) of the LOG_CONNECTION MTA option was set, SMTP auth was enabled, or both. Additionally, evaluation only occurred when an AUTH, EHLO, or HELO command was issued. This has now been changed; PORT_ACCESS is now evaluated unconditionally as soon as the SMTP server thread starts, before the banner is sent.

PORT_ACCESS Mapping Flags

The following flags and metacharacters apply to the PORT_ACCESS mapping table. The SEND_ACCESS, ORIG_SEND_ACCESS, MAIL_ACCESS, and ORIG_MAIL_ACCESS and FROM_ACCESS Mapping Table have different sets of flags.

- PORT_ACCESS Input Flags
- PORT_ACCESS Output Flags
- PORT_ACCESS Table Examples

See Mappings File for more details about the format and operation, and patterns, templates, and

metacharacters which apply to all mapping tables.

PORT_ACCESS Input Flags

Input flags are set by the process that calls the mapping table and tested for by using the `$:` and `$;` metacharacters in the template. The `$:` will succeed if the flag is set or fail if it is not set. The `$;` will succeed if the flag is not set or fail if it is set. The following table shows the `$:`*x* form to test if the flag is set. The `$;`*x* form can also be used to test if condition is not true.

The `PORT_ACCESS` table is consulted by both the `dispatcher` and the `tcp_smtp_server` or `tcp_lmtp_server` processes. Mapping table entries that perform callouts to external functions that may incur some overhead should be done in one or the other, but not both.

Flag	Description
$:A	Match only when the probe is performed by the Dispatcher
$:S	Match only when the probe is performed by an SMTP server or LMTP server

See To Use DNS Lookups Including RBL Checking for SMTP Relay Blocking for examples of using `$:A`, `$;A`, `$:S`, and `$;S`.

PORT_ACCESS Output Flags

Output flags are set by the template in the mapping table entry and consumed upon return to the calling program. The *Valid* column in the tables below indicates whether the flag is used by `dispatcher` (D), `tcp_smtp_server` (S), or `tcp_lmtp_server` (L). The A or S input flags should be used to ensure that rules are only applied in the appropriate context.

Flag	Valid	Description
$U	S	Enable channel (slave) debugging (This feature was introduced in **Messaging Server 6.3.); if `MM_DEBUG` or `OS_DEBUG` are set, enable those as well (This feature was introduced in *Messaging Server 7u3.**
$V	S	Enable the MTA's private SMTP extensions XADR, XCIR, XGEN, and XSTA, overriding any SMTP server `DISABLE_` **channel options. This feature was introduced in *Messaging Server 7.**
$/	DS	Set the "fast disconnect" flag for sessions that have not yet succeeded in starting a transaction; for such sessions, any subsequent disconnect is done with SO_LINGER enabled and a timeout of 0, which may clear slots quicker on intermediate firewalls and proxies. This feature was introduced in **Messaging Server 7.**
$Y	DSL	Allow access.
$T	D	If bit 1 (value 2) of the `LOG_CONNECTION` MTA option is set and the $N flag is set so that the connection is rejected, then $T outputs the entire right hand side text in a "T" (or "X" ++) record. The T log entry will include the entire mapping result string ($N and its string). In contrast, bit 4 of `LOG_CONNECTION` is a different effect: it will cause material after two vertical bars to be included in normal "C" (connection close) records.

Flags with Arguments, in Output Flag Argument Order+ (DO NOT ALPHABETIZE THIS LIST!)

Flag	Valid	Description
$<*string*	DSL	Send *string* to syslog if probe matches.
$>*string*	DSL	Send *string* to syslog if access is rejected.
$N*string* $F*string*	DSL	Reject access with the optional error text *string*. $F is a synonym for $N
SASL Ruleset	S	Not used, but you must enter an empty value (double bar with no space, "||") if you want to use any of the following flags.
SASL Realm	S	Not used, but you must enter an empty value (double bar with no space, "||") if you want to use any of the following flags.
Application Info	SL	If the LOG_CONNECTION MTA option is set to bit 4 (value 16), PORT_ACCESS is allowed to add text to application information string. This is where the string can be specified. If it is not used, you must enter an empty value (double bar with no space, "||") if you want to use any of the flags below.
$D*delay*	S	Specifies the number of centiseconds to delay before purging and sending the banner. A value of 0 disabled both the delay and purge. This value has the same semantics as the BANNER_PURGE_DELAY value. Note that any PORT_ACCESS mapping setting overrides the BANNER_PURGE_DELAY SMTP channel option. See Anti-Spam Technique: Delay Sending the SMTP Banner for details on using this anti-spam feature.
$S*channel*	S	Set *channel* as the source channel for this SMTP session. This feature was introduced in **Messaging Server 7.**
Certificate Nickname	S	Comma-separated list of Server Certificates to send to client for TLS

{+} To use multiple flags with arguments, separate the arguments with the vertical bar character, |, placing the arguments in the order listed in this table.
++ The T record discussed in the $T flag above occurs when the PORT_ACCESS table entry is being evaluated by the dispatcher process. If it is evaluated by the tcp_smtp_server it will produce an X record instead.

PORT_ACCESS Table Examples

For example, the following mapping will only accept SMTP connections (to port 25, the normal SMTP port) from a single network, except for a particular host singled out for rejection without explanatory text:

```
PORT_ACCESS

  TCP|*|25|192.123.10.70|*  $N500
  TCP|*|25|192.123.10.*|*   $Y
  TCP|*|25|*|*              $N500$ Bzzzt$ thank$ you$ for$ playing.
```

You need to restart the Dispatcher after making any changes to the PORT_ACCESS mapping table so that the Dispatcher will see the changes. (If you are using a compiled MTA configuration, you will first need to recompile your configuration using the imsimta cnbuild command to get the change incorporated into the compiled configuration.)

The PORT_ACCESS mapping table is specifically intended for performing IP-based rejections. For more general control at the email address level, the SEND_ACCESS or MAIL_ACCESS mapping table, might be more appropriate.

PORT_ACCESS Output Flags lists the order in which arguments are consumed from the template for certain flags and other arguments that can be returned from the PORT_ACCESS table.

```
PORT_ACCESS

  *|*|*|*|*  $C$|INTERNAL_IP;$3|$Y$E
  *  $YEXTERNAL
```

In this default configuration, it appears as if the $Y is being given a mysterious "EXTERNAL" argument. But $Y does not take any arguments. That "EXTERNAL" is actually the value for the SASL Ruleset argument. If other argument-consuming flags were specified, you would use "|" to separate the arguments.

SASL Ruleset

If you set a ruleset, then configutil parameters of the form `sasl.<ruleset>.ldap.<option-name>` apply.

Note that the PORT_ACCESS mapping supplied by the initial configuration does set the ruleset to "EXTERNAL" for incoming connections that don't match in INTERNAL_IP – that way you can, in theory, configure different SASL options for the "internal" connections (in the "DEFAULT" ruleset) vs. the "external" connections (in the "EXTERNAL" ruleset).

SASL Realm

Realm is used in place of `service.defaultdomain` if the userid supplied for authentication is unqualified.

Application Info

If the LOG_CONNECTION MTA option is set to bit 4 (value 16), PORT_ACCESS is allowed to add text to application information string. This is where the string can be specified. If the option is enabled but you do not want to specify any additional app info in a rule, you must enter an empty value (double bar with no space, "||") if you want to use any of the later flags/arguments.

Certificate Nickname

```
PORT_ACCESS

  *|*|*|*|*  $C$|INTERNAL_IP;$3|$Y$E
  TCP|1.2.3.4|*|*|* $;A$Y||xyzzy
  *  $YEXTERNAL
```

In this example, if the server has multiple IP addresses, for connections from clients to the server's IP address 1.2.3.4, the `tcp_smtp_server` process will use the Certificate with the nickname `xyzzy` for TLS instead of the default certificate. The "||" supplies null values for Ruleset and Realm. If the LOG_CONNECTION MTA option is set to bit 4 (value 16), an additional "|" would be needed, for example:

```
PORT_ACCESS

  *|*|*|*|*  $C$|INTERNAL_IP;$3|$Y$E
  TCP|1.2.3.4|*|*|* $;A$Y|||xyzzy
  *  $YEXTERNAL
```

IP_ACCESS Mapping Table

The IP_ACCESS Mapping Table can be used to do a last moment check on the IP address to which the MTA is about to connect. The connection attempt can then be aborted or redirected. This can be useful under certain special circumstances, for example, security concerns about a destination IP address to which should never be connected, or where it is wished to avoid connecting to known-to-be-bogus destination IP addresses (for example, 127.0.0.1), or where you wish to attempt to fail over to another destination IP address similar to a `lastresort` keyword effect (see Configuring Channel Definitions).

This access mapping is consulted during SMTP client operations just prior to attempting to open connections to a remote server. The mapping probe has the following format:

```
source-channel|address-current|address-count|ip-current|hostname
```

`source-channel` is the channel from which the message is being dequeued. `address-count` is the total number of IP addresses for the remote server. `address-current` is the index of the current IP address being tried. `ip-current` is the current IP address. `hostname` is the symbolic name of the remote server. The following table shows the flags for this table.

IP_ACCESS Mapping Table Flags

The following flags apply to the IP_ACCESS mapping table. See Mappings File for more details about the format and operation, and patterns, templates, and metacharacters which apply to all mapping tables.

Flag	Description
$N	Immediately reject the message with an "invalid host/domain error." Any supplied text will be logged as the reason for rejection but will not be included in the DSN.
$I	Skip the current IP without attempting to connect.
$A	Replace the current IP address with the mapping result.

When Access Controls Are Applied

Messaging Server checks access control mappings as early as possible. Exactly when this happens depends upon the email protocol in use – when the information that must be checked becomes available.

For the SMTP protocol, a `FROM_ACCESS` rejection occurs in response to the MAIL FROM: command, before the sending side can send the recipient information or the message data. A `SEND_ACCESS` or `MAIL_ACCESS` rejection occurs in response to the RCPT TO: command, before the sending side gets to send the message data. If an SMTP message is rejected, Messaging Server never accepts or sees the message data, thus minimizing the overhead of performing such rejections.

If multiple access control mapping tables exist, Messaging Server checks them all. That is, a `FROM_ACCESS`, a `SEND_ACCESS`, an `ORIG_SEND_ACCESS`, a `MAIL_ACCESS`, and `ORIG_MAIL_ACCESS` mapping tables may all be in effect.

`PORT_ACCESS` is called from dispatcher as soon as it accepts the incoming TCP connection. It is also called from `tcp_smtp_server` when any of the `maysaslserver` or `mustsaslserver` keywords are present on the source channel. (See Configuring Channel Definitions.)

`FROM_ACCESS` is used by the `tcp_smtp_server` when processing the `MAIL FROM SMTP` command.

`SEND_ACCESS` and `ORIG_SEND_ACCESS` tables are used by the `tcp_smtp_server` when processing the `RCPT TO SMTP` command.

MAIL_ACCESS and ORIG_MAIL_ACCESS tables are used by the tcp_smtp_server when processing the RCPT TO SMTP command.

To Test Access Control Mappings

The imsimta test -rewrite utility – particularly with the -from, -source_channel, -sender and -destination_channel options – can be useful in testing access control mappings. See the imsimta test documentation in *Messaging Server Administration Reference* for details. The example below shows a sample SEND_ACCESS mapping table and the resulting probe.

MAPPING TABLE:

```
ORIG_SEND_ACCESS

  tcp_local|friendly@siroe.com|*|User@sesta.com     $Y
  tcp_local|unwelcome@varrius.com|*|User@sesta.com  $NGo$ away!
```

PROBE:

```
$ imsimta test -rewrite -from="friendly@siroe.com" -source=tcp_local
User@sesta.com
...
Submitted address list:
  ims-ms
    user@ims-ms-daemon (orig User@sesta.com, inter User@sesta.com, host
ims-ms-daemon)
*NOTIFY-FAILURES* *NOTIFY-DELAYS*

Submitted notifications list:

$ imsimta test -rewrite -from="unwelcome@varrius.com" -source=tcp_local
User@sesta.com
...
Submitted address list:
Address list error -- 5.7.1 Go away!: User@sesta.com

Submitted notifications list:
```

To Limit Specified IP Address Connections to the MTA

To limit how often a particular IP address can connect to the MTA, see [Using and Configuring MeterMaid for
Access Control]. Limiting connections by particular IP addresses can be useful for preventing excessive connections used in denial-of-service attacks. In the past, this function was performed using the shared library, conn_throttle.so in the Port Access mapping table. No new enhancements are planned for conn_throttle.so and MeterMaid is its more effective replacement.

conn_throttle.so is a shared library used in a PORT_ACCESS mapping table to limit MTA connections made too frequently from particular IP addresses. All configuration options are specified as parameters to the connection throttle shared library as follows:

$[*msg-svr-base*/lib/conn_throttle.so,throttle,*IP-address*,*max-rate*]

IP-address is the dotted-decimal address of the remote system. *max-rate* is the connections per minute that shall be the enforced maximum rate for this IP-address.

The routine name `throttle_p` may be used instead of `throttle` for a penalizing version of the routine. `throttle_p` will deny connections in the future if they've connected too many times in the past. If the maximum rate is 100, and 250 connections have been attempted in the past minute, not only will the remote site be blocked after the first 100 connections in that minute, but they'll also be blocked during the second minute. In other words, after each minute, max-rate is deducted from the total number of connections attempted and the remote system is blocked as long as the total number of connections is greater than the maximum rate.

If the IP-address specified has not exceeded the maximum connections per minute rate, the shared library callout will fail.

If the rate has been exceeded, the callout will succeed, but will return nothing. This is done in a $C/$E combination as in the example:

```
PORT_ACCESS

  TCP|*|25|*|* \
$C$[_msg-svr-base_/lib/conn_throttle.so,throttle,$1,10] \
$N421$ Connection$ not$ accepted$ at$ this$ time$E
```

Where,

`$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.

`$[`*msg-svr-base*`/lib/conn_throttle.so,throttle,$1,10]` is the library call with `throttle` as the library routine, `$1` as the server IP Address, and 10 the connections per minute threshold.

`$N421$ Connection$ not$ accepted$ at$ this$ time` rejects access and returns the 421 SMTP code (transient negative completion) along with the message "Connection not accepted at this time".

`$E` ends the mapping process now. It uses the output string from this entry as the final result of the mapping process.

To Add SMTP Relaying

Messaging Server is, by default, configured to block attempted SMTP relays. That is, it rejects attempted message submissions to external addresses from unauthenticated external sources (external systems are any other system than the host on which the server itself resides). This default configuration is quite aggressive in blocking SMTP relaying in that it considers all other systems to be external systems.

IMAP and POP clients that attempt to submit messages via the Messaging Server system's SMTP server destined for external addresses, and who do not authenticate using SMTP AUTH (SASL), will find their submission attempts rejected. Thus, you will likely want to modify your configuration so that it recognizes your own internal systems and subnets from which relaying should always be accepted.

Which systems and subnets are recognized as internal is normally controlled by the `INTERNAL_IP` mapping table, which is located in the *msg-svr-base*`/config/mappings` directory.

For instance, on a Messaging Server whose IP address is 123.45.67.89, the default `INTERNAL_IP` mapping table would appear as follows:

```
INTERNAL_IP

   $(123.45.67.89/32)   $Y
   127.0.0.1            $Y
   *   $N
```

Here the initial entry, using the `$(IP-pattern/signicant-prefix-bits)` syntax, is specifying that any IP address that matches all 32 bits of 123.45.67.89 should match and be considered internal. The second entry recognizes the loopback IP address 127.0.0.1 as internal. The final entry specifies that all other IP addresses should not be considered internal. All entries must be preceded by at least one space.

You add additional entries by specifying additional IP addresses or subnets before the final $N entry. These entries must specify an IP address or subnet (using the `$(.../...)` syntax to specify a subnet) on the left side and $Y on the right side. Or you may modify the existing `$(.../...)` entry to accept a more general subnet.

For instance, if this same sample site has a class-C network, that is, it owns all of the 123.45.67.0 subnet, then the site would want to modify the initial entry by changing the number of bits used in matching the address. In the mapping table below, we change from 32 bits to 24 bits. This allows all clients on the class-C network to relay mail through this SMTP relay server.

```
INTERNAL_IP

   $(123.45.67.89/24)   $Y
   127.0.0.1   $Y
   *   $N
```

Or if the site owns only those IP addresses in the range 123.45.67.80-123.45.67.99, then the site would want to use:

```
INTERNAL_IP

! Match IP addresses in the range 123.45.67.80-123.45.67.95
   $(123.45.67.80/28) $Y
! Match IP addresses in the range 123.45.67.96-123.45.67.99
   $(123.45.67.96/30) $Y
   127.0.0.1 $Y
   * $N
```

Note that the `imsimta test -match` utility can be useful for checking whether an IP address matches a particular `$(.../...)` test condition. The `imsimta test -mapping` utility can be more generally useful in checking that your `INTERNAL_IP` mapping table returns the desired results for various IP address inputs.

After modifying your `INTERNAL_IP` mapping table, be sure to issue the `imsimta restart` command (if you are not running with a compiled configuration) or an `imsimta cnbuild` followed by an `imsimta restart smtp` (if you are running with a compiled configuration) so that the changes take effect.

Further information on the mapping file and general mapping table format, as well as information on `imsimta` command-line utilities, can be found in *Messaging Server Administration Reference*.

Allowing SMTP Relaying for External Sites

All internal IP addresses should be added to the INTERNAL_IP mapping table as discussed above. If you have friendly or companion systems/sites from which you wish to allow SMTP relaying, the simplest approach is to include them along with your true internal IP addresses in your INTERNAL_IP mapping table.

If you don't wish to consider these as true internal systems/sites, (for instance, if for logging or other control purposes you wish to distinguish between *true internal systems* versus the *friendly non-internal systems with relay privileges*), there are other ways to configure the system.

One approach is to set up a special channel for receiving messages from such friendly systems. Do this by creating a *tcp_friendly* channel akin to your existing *tcp_internal* channel with official host name *tcp_friendly-daemon*, and a FRIENDLY_IP mapping table akin to your INTERNAL_IP mapping table that lists the friendly system IP addresses. Then right after the current rewrite rule:

```
! Do mapping lookup for internal IP addresses
[]    $E$R${INTERNAL_IP,$L}$U%[$L]@tcp_intranet-daemon
```

add a new rewrite rule:

```
! Do mapping lookup for "friendly", non-internal IP addresses
[]     $E$R${FRIENDLY_IP,$L}$U%[$L]@tcp_friendly-daemon
```

An alternate approach is to add to your ORIG_SEND_ACCESS mapping table above the final $N entry, new entries of the form:

```
tcp_local|*@siroe.com|tcp_local|*     $Y
```

where siroe.com is the name of a friendly domain, and to add an ORIG_MAIL_ACCESS mapping table of the form:

```
ORIG_MAIL_ACCESS

   TCP|*|25|$(match-siroe.com-IP-addresses)|*|SMTP*|MAIL|   \
tcp_local|*@siroe.com|tcp_local|*      $Y
   TCP|*|*|*|*|SMTP*|MAIL|tcp_local|*|tcp_local|* $N
```

where the $(...) IP address syntax is the same syntax described in the previous section. The ORIG_SEND_ACCESS check will succeed as long as the address is ok, so we can go ahead and also do the ORIG_MAIL_ACCESS check which is more stringent and will only succeed if the IP address also corresponds to an siroe.com IP address.

Configuring SMTP Relay Blocking

You can use access control mappings to prevent people from relaying SMTP mail through your Messaging Server system. For example, you can prevent people from using your mail system to relay junk mail to hundreds or thousands of Internet mailboxes.

By default, Messaging Server prevents all SMTP relaying activity, including relaying by local POP and IMAP users.

Blocking unauthorized relaying while allowing it for legitimate local users requires configuring Messaging

Server to know how to distinguish between the two classes of users. For example, local users using POP or IMAP depend upon Messaging Server to act as an SMTP relay.

To prevent SMTP relay, you must be able to:

- Differentiate Between Internal and External Mail
- Differentiate Authenticated Users' Mail
- Prevent Mail Relay

To enable SMTP relay by internal hosts and clients, you must add your "internal" IP addresses or subnets to the `INTERNAL_IP` mapping table.

How the MTA Differentiates Between Internal and External Mail

To block mail relaying activities, the MTA must first be able to differentiate between internal mail originated at your site and external mail originated out on the Internet and passing through your system back out to the Internet. The former class of mail you want to permit; the latter class you want to block. This differentiation is achieved using the `switchchannel` keyword on your inbound SMTP channel, usually the `tcp_local` channel, and is set by default.

The `switchchannel` keyword works by causing the SMTP server to look at the actual IP address associated with the incoming SMTP connection. Messaging Server uses that IP address, in conjunction with your rewrite rules, to differentiate between an SMTP connection originated within your domain and a connection from outside of your domain. This information can then be used to segregate the message traffic between internal and external traffic.

The MTA configuration described below is setup by default so that the server can differentiate between your internal and external message traffic.

- In the configuration file, immediately before the local channel, is a `defaults` channel with the `noswitchchannel` keyword:

  ```
  defaults notices 1 2 4 7 noswitchchannel immnonurgent maxjobs 7
  defaulthost siroe.com siroe.com
  ```

- The incoming TCP/IP channel specifies the `switchchannel` and `remotehost` keywords; for example:

  ```
  tcp_local smtp single_sys mx switchchannel remotehost
  tcp-daemon
  ```

- After the incoming TCP/IP channel definition, is a similar channel with a different name which specifies the `allowswitchchannel` keyword; for example:

  ```
  tcp_intranet smtp single_sys mx allowswitchchannel
  tcp_intranet-daemon
  ```

  With the above configuration settings, SMTP mail generated within your domain will come in via the `tcp_intranet` channel. All other SMTP mail will come in via the `tcp_local` channel. Mail is distinguished between internal and external based upon which channel it comes in on.

How does this work? The key is the `switchchannel` keyword. The keyword is applied to the `tcp_local` channel. When a message comes in your SMTP server, that keyword causes the server to look at the source IP address associated with the incoming connection. The server attempts a reverse-pointing envelope rewrite of the literal IP address of the incoming connection, looking for an associated channel. If the source IP address matches an IP address or subnet in your `INTERNAL_IP`

mapping table, the rewrite rule which calls out to that mapping table causes the address to rewrite to the `tcp_intranet` channel.

Since the `tcp_intranet` channel is marked with the `allowswitchchannel` keyword, the message is switched to the `tcp_intranet` channel and comes in on that channel. If the message comes in from a system whose IP address is not in the `INTERNAL_IP` mapping table, the reverse-pointing envelope rewrite will either rewrite to the `tcp_local` or, perhaps to some other channel. However, it will not rewrite to the `tcp_intranet` channel and since all other channels are marked `noswitchchannel` by default, the message will not switch to another channel and will remain with the `tcp_local` channel.

> **Note**
> Any mapping table or conversion file entries which use the string "`tcp_local`" may need to be changed to either "`tcp_*`" or "`tcp_intranet`" depending upon the usage.

Differentiate Authenticated Users' Mail

Your site might have "local" client users who are not part of your physical network. When these users submit mail, the message submissions come in from an external IP address, for instance, arbitrary Internet Service Providers. If your users use mail clients that can perform SASL authentication, then their authenticated connections can be distinguished from arbitrary other external connections. The authenticated submissions you can then permit, while denying non-authenticated relay submission attempts. Differentiating between authenticated and non-authenticated connections is achieved using the `saslswitchchannel` keyword on your inbound SMTP channel, usually the `tcp_local` channel.

The `saslswitchchannel` keyword takes an argument specifying the channel to switch to; if an SMTP sender succeeds in authenticating, then their submitted messages are considered to come in the specified switched to channel.

To Add Distinguishing Authenticated Submissions

1. In your configuration file, add a new TCP/IP channel definition with a distinct name; for example:

   ```
   tcp_auth smtp mustsaslserver noswitchchannel
   tcp_auth-daemon
   ```

   This channel should not allow regular channel switching (that is, it should have `noswitchchannel` on it either explicitly or implied by a prior defaults line). This channel should have `mustsaslserver` on it.

2. Modify your `tcp_local` channel by adding `maysaslserver` and `saslswitchchannel tcp_auth`, as shown in the following example:

   ```
   tcp_local smtp mx single_sys maysaslserver saslswitchchannel
   tcp_auth switchchannel
   tcp-daemon
   ```

   With this configuration, SMTP mail sent by users who can authenticate with a local password will now come in the `tcp_auth` channel. Unauthenticated SMTP mail sent from internal hosts will still come in `tcp_internal`. All other SMTP mail will come in `tcp_local`.

Prevent Mail Relay

Now to the point of this example: preventing unauthorized people from relaying SMTP mail through your

system. First, keep in mind that you want to allow local users to relay SMTP mail. For instance, POP and IMAP users rely upon using Messaging Server to send their mail. Note that local users may either be physically local, in which case their messages come in from an internal IP address, or may be physically remote but able to authenticate themselves as local users.

You want to prevent random people out on the Internet from using your server as a relay. With the configuration described in the following sections, you can differentiate between these classes of users and block the correct class. Specifically, you want to block mail from coming in your `tcp_local` channel and going back out that same channel. To that end, an `ORIG_SEND_ACCESS` mapping table is used.

An `ORIG_SEND_ACCESS` mapping table may be used to block traffic based upon the source and destination channel. In this case, traffic from and back to the `tcp_local` channel is to be blocked. This is realized with the following `ORIG_SEND_ACCESS` mapping table:

```
ORIG_SEND_ACCESS

  tcp_local|*|tcp_local|*  $N$D30|Relaying$ not$ allowed
```

In this example, the entry states that messages cannot come in the `tcp_local` channel and go right back out it. That is, this entry disallows external mail from coming in your SMTP server and being relayed right back out to the Internet.

An `ORIG_SEND_ACCESS` mapping table is used rather than a `SEND_ACCESS` mapping table so that the blocking will not apply to addresses that originally match the ims-ms channel (but which may expand via an alias or mailing list definition back to an external address). With a `SEND_ACCESS` mapping table one would have to go to extra lengths to allow outsiders to send to mailing lists that expand back out to external users, or to send to users who forward their messages back out to external addresses.

To Use DNS Lookups Including RBL Checking for SMTP Relay Blocking

In the Messaging Server, there are a number of different ways to ensure that all mail accepted for delivery or forwarding comes from an address with a valid DNS name. The simplest way is to put the `mailfromdnsverify` channel keyword on the `tcp_local` channel.

Messaging Server also provides the `dns_verify` program which allows you to ensure that all mail accepted for delivery or forwarding comes from an address with a valid DNS name using the following rule in `ORIG_MAIL_ACCESS`:

```
ORIG_MAIL_ACCESS

  TCP|*|*|*|*|SMTP*|MAIL|*|*@*|*|*  \
$[_msg-svr-base_/lib/dns_verify.so,\
dns_verify,$7|$$y|$$NInvalid$ host:$ $$7$ -$ %e]
```

The line breaks in the above example are syntactically significant in such mapping entries. The backslash character is a way of legally continuing on to the next line.

The `dns_verify` image can also be used to check incoming connections against things like the RBL (Realtime Blackhole List), MAPS (Mail Abuse Prevention System, DUL (Dial-up User List), or ORBS (Open Relay Behavior-modification System) lists as another attempt to protect against UBE. As with the new `mailfromdnsverify` keyword, there's also a separate "simpler to configure" approach one can use for such checks rather than doing the `dns_verify` callout. The simpler approach is to use the `DNS_VERIFY_DOMAIN` option in the `dispatcher.cnf` file. For example, in the `[SERVICE=SMTP]` section, set instances of the option to the various lists you want to check against:

```
[SERVICE=SMTP]
PORT=25
! ...rest of normal options...
DNS_VERIFY_DOMAIN=sbl-xbl.spamhaus.org.
DNS_VERIFY_DOMAIN=list.dsbl.org.
...etc...
```

In this case, messages are rejected at the SMTP level, that is, the messages are rejected during the SMTP dialogue and thus never sent to the MTA. The disadvantage of this simpler approach is that it does the checks for all normal incoming SMTP messages including those from internal users. This is less efficient and potentially problematic if your Internet connectivity goes down. An alternative is to call out to `dns_verify` from a `PORT_ACCESS` mapping table or `ORIG_MAIL_ACCESS` mapping table. In the `PORT_ACCESS` mapping table, you can have an initial entry or entries that don't check for local internal IP addresses or message submitters and a later entry that does the desired check for everyone else. Or, in an `ORIG_MAIL_ACCESS` mapping table, if you only apply the check on messages coming in the tcp_local channel then you're skipping it for messages coming from your internal systems/clients. Examples using the entry points to `dns_verify` are shown below.

```
PORT_ACCESS

! Allow internal connections in unconditionally
  *|*|*|*|* $C$|INTERNAL_IP;$3|$Y$E
! Check other connections against RBL list
  TCP|*|25|*|* \
$C$[_msg-svr-base_/lib/dns_verify.so,dns_verify_domain_port,$1,\
dnsblock.siroe.com.,Your$ host$ ($1)$ found$ on$ dnsblock$ list]$E
  * $YEXTERNAL

ORIG_MAIL_ACCESS

  TCP|*|25|*|*|SMTP*|*|tcp_local|*@*|*|*  \
$C$[_msg-svr-base_/lib/dns_verify.so,\
dns_verify_domain,$1,sbl-xbl.spamhaus.org.]$E
```

For more information see: Performance Tuning DNS Realtime BlockLists (RBL) Lookups.

The `PORT_ACCESS` table is probed both by the dispatcher, when accepting connections, and by the `tcp_smtp_server` process under certain circumstances.

The `tcp_smtp_server` processes always check the `PORT_ACCESS` table for channels marked `maysaslserver` or `mustsaslserver`, and they will do it for all channels if bit 4 (value 16) of the `LOG_CONNECTION` in `option.dat` is set.

So if either of those are true, the customer needs to be aware that his `PORT_ACCESS` table is being processed twice for every connection. This may be a trivial overhead except that callouts to things like DNS RBLs or LDAP lookups can be relatively expensive in terms of their impact on SMTP server response time. For this reason, you want to avoid doing them any more than necessary. That is one reason the callout in the example above is coded after the `INTERNAL_IP` lookup. If the SMTP client is in your `INTERNAL_IP` table, then you allow the connection without doing the RBL lookup. Reversing that order would mean your local clients would be delayed by RBL lookups. To prevent doing this twice, add a check for one the flags set by dispatcher or `tcp_smtp_server` so that you only process that table entry when one of those is set or not. For example, add `$:A` to the entry:

```
TCP|*|25|*|*   \
$C$:A$[_msg-svr-base_/lib/dns_verify.so,dns_verify_domain_port,$1,\
dnsblock.siroe.com.,Your$ host$ ($1)$ found$ on$ dnsblock$ list]$E
    * $YEXTERNAL
```

We added the `$:A` after the `$C` so the table processing will continue down the table if this does not match. The `$:A` specifies that this entry be processed only if it is being done by the dispatcher, that is, not when done by `tcp_smtp_server`. Alternatively, if you wanted to cause it to be done only in `tcp_smtp_server`, use `$:S` instead:

```
TCP|*|25|*|*   \
$C$:S$[_msg-svr-base_/lib/dns_verify.so,dns_verify_domain_port,$1,\
dnsblock.siroe.com.,Your$ host$ ($1)$ found$ on$ dnsblock$ list]$E
    * $YEXTERNAL
```

The negative check would be `$;A` to check that `A` is not set, or `$;S` to check that `S` is not set.

Support for DNS-based Databases

The `dns_verify` program supports DNS-based databases used to determine incoming SMTP connections that might send unsolicited bulk mail. Some of the publicly available DNS databases do not contain TXT records that are typically used for this purpose. Instead, they only contain A records.

In a typical setup, the TXT record found in the DNS for a particular IP address contains an error message suitable to return to the SMTP client when refusing a message. But, if a TXT record is not found and an A record is found, then versions of `dns_verify` prior to Messaging Server 5.2 returned the message "No error text available."

`dns_verify` now supports an option that specifies a default text that is used in the event that no TXT record is available. For example, the following PORT_ACCESS mapping table shows how to enable this option:

```
PORT_ACCESS

   *|*|*|*|* $C$|INTERNAL_IP;$3|$Y$E
   TCP|*|25|*|*   \
$C$[_msg-svr-base_/lib/dns_verify.so \
,dns_verify_domain_port,$1,dnsblock.siroe.com,Your$ host$ ($1)$ \
found$ on$ dnsblock$ list]$E
    * $YEXTERNAL
```

In this example, if the remote system is found in a query in the domain `dnsblock.siroe.com`, but no TXT record is available, then the following message is returned, "Your host a.b.c.d found on dnsblock list."

Handling Large Numbers of Access Entries

Sites that use very large numbers of entries in mapping tables should consider organizing their mapping tables to have a few general wildcarded entries that call out to the general text database for the specific lookups. It is much more efficient to have a few mapping table entries calling out to the general text database for specific lookups than to have huge numbers of entries directly in the mapping table.

One case in particular is that some sites like to have per user controls on who can send and receive Internet email. Such controls are conveniently implemented using an access mapping table such as ORIG_SEND_ACCESS. For such uses, efficiency and performance can be greatly improved by storing the bulk of the specific information (that is, specific addresses) in the general text database with mapping table entries structured to call out appropriately to the general text database.

For example, consider the ORIG_SEND_ACCESS mapping table shown below.

```
ORIG_SEND_ACCESS

! Users allowed to send to Internet
!
  *|adam@siroe.com|tcp_local|*    $Y
  *|betty@siroe.com|tcp_local|*   $Y
! ...etc...
!
! Users not allowed to send to Internet
!
  *|norman@siroe.com|tcp_local|*  $NInternet$ access$ not$ permitted
  *|opal@siroe.com|tcp_local|*    $NInternet$ access$ not$ permitted
! ...etc...
!
! Users allowed to receive from the Internet
!
  tcp_*|*|*|adam@siroe.com        $Y
  tcp_*|*|*|betty@siroe.com       $Y
! ...etc...
!
! Users not allowed to receive from the Internet
!
  tcp_*|*|*|norman@siroe.com      $NInternet$ e-mail$ not$ accepted
  tcp_*|*|*|opal@siroe.com        $NInternet$ e-mail$ not$ accepted
! ...etc...
```

Rather than using such a mapping table with each user individually entered into the table, a more efficient setup (much more efficient if hundreds or thousands of user entries are involved) is shown in the example below, which shows sample source text file for a general database and a sample ORIG_SEND_ACCESS mapping table. See MTA Text Databases for set up information.

DATABASE ENTRIES

```
SEND|adam@domain.com    $Y
SEND|betty@domain.com   $Y
! ...etc...
SEND|norman@domain.com  $NInternet$ access$ not$ permitted
SEND|opal@domain.com    $NInternet$ access$ not$ permitted
! ...etc...
RECV|adam@domain.com    $Y
RECV|betty@domain.com   $Y
! ...etc...
RECV|norman@domain.com  $NInternet$ e-mail$ not$ accepted
RECV|opal@domain.com    $NInternet$ e-mail$ not$ accepted
```

MAPPING TABLE

```
ORIG_SEND_ACCESS

! Check if may send to Internet
!
  *|*|*|tcp_local       $C${SEND|$1}$E
!
! Check if may receive from Internet
!
  tcp_*|*|*|*           $C${RECV|$3}$E
```

In this example, the use of the arbitrary strings `SEND|` and `RECV|` in the general database left-hand sides (and hence in the general database probes generated by the mapping table) provides a way to distinguish between the two sorts of probes being made. The wrapping of the general text database probes with the `$C` and `$E` flags, as shown, is typical of mapping table callouts to the general text database.

The above example showed a case of simple mapping table probes getting checked against general text database entries. Mapping tables with much more complex probes can also benefit from use of the general text database.

Controlling the Envelope From: Address in Mappings Strings and Mailing List Named Parameters and LDAP Attributes

The `use_auth_return`, `use_canonical_return`, and `use_orig_return MTA` options control which envelope address to use in certain mapping tables or mailing list named parameters (if using the aliases file for mailing lists) or attributes (if using LDAP mailing lists).

For more information on these options, see use_auth_return, use_canonical_return, use_orig_return MTA Options.

PART 2. MAILBOX FILTERS

Mailbox filters, also called Sieve filters, filter messages containing specified strings found in the message headers and apply specified actions to these mail message. Administrators can filter mail streams going to a user, through a channel, or through an MTA. Messaging Server filters are stored on the server and evaluated by the server, hence, they are sometimes called server-side rules (SSR).

This section contains the following topics:

- Sieve Filter Support
- Sieve Filtering Overview
- To Create User-level Filters
- To Create Channel-level Filters
- To Create MTA-Wide Filters
- To Debug User-level Filters

Sieve Filter Support

Messaging Server filters are based on the Sieve filtering language, Draft 9 of the Sieve Internet Draft. See RFC3028 for more information about Sieve syntax and semantics. In addition, Messaging Server also supports the following Sieve extensions:

- ***jettison***. Similar to `discard` in that it causes messages to be silently discarded, but unlike

discard, which does nothing but cancel the implicit keep, jettison forces a discard to be performed. The behavioral difference is only relevant when multiple Sieve filters are involved. For example, a system level discard can be overridden by a user Sieve filter explicitly specifying keep, whereas a system level jettison will override anything done by a user Sieve.

- ***Head-of-household Sieve filters***. Provides a means by which one user can specify a Sieve filter for another user. Uses two LDAP attributes in a user entry controlled by these MTA options:
  - LDAP_PARENTAL_CONTROLS - Specifies an attribute containing a string value of either Yes or No. Yes means a head of household Sieve is to be applied to this entry, No means no such Sieve is to be applied. No default.
  - LDAP_FILTER_REFERENCE - Specifies an attribute containing a DN pointing to a directory entry where the head of household Sieve can be found. No default.
    The entry containing the head of household Sieve must contain two attributes specified by the following MTA options:
  - LDAP_HOH_FILTER - Specifies an attribute containing the head of household Sieve. The value of this option defaults to mailSieveRuleSource.
  - LDAP_HOH_OWNER - Specifies an attribute containing the email address of the owner of the head of household. The value of this option defaults to mail.
    Both of these attributes must be present for the head of household Sieve to work.
- Sieve redirect can now add three header fields:

```
resent-date: _date-of-resend-operation_
resent-to: _address-specified-in-redirect_
resent-from: _addres-of-sieve-owner_
```

  The new :resent and :noresent arguments to redirect can be used to control whether or not these fields are added. If neither argument is specific the system default is used. The system default is controlled by the new SIEVE_REDIRECT_ADD_RESENT MTA option. Setting the option to 1 causes these fields to be generated unless :noresent used. A setting of 0 causes the fields to be generated only if :resent is used. The option defaults to 1, which means the fields are generated by default for regular redirects.

- Sieve redirect has been enhanced with three new arguments:
  :resetmailfrom - Reset the envelope FROM: address to that of the current Sieve owner.
  :keepmailfrom - Preserve the envelope FROM: address from the original message.
  :notify - Specify a new set of notification flags for the redirected message. A single parameter is required giving a list of notification flags. The same set of flags accepted by the NOTIFY parameter of the DSN SMTP extension are accepted here: SUCCESS, FAILURE, DELAY and NEVER. Note that these flags are specified as a Sieve list, for example:

```
redirect :notify ["SUCCESS","FAILURE"] "foo@example.com";
```

  The default if :notify isn't specified as the normal SMTP default of FAILURE, DELAY. :keepmailfrom is the default unless :notify is specified, in which case the default switches to :resetmailfrom. The one additional exception is that specification of the SUCCESS flag forces the use of :resetmailfrom unconditionally.

Sieve Filtering Overview

A Sieve filter consists of one or more conditional actions to apply to a mail message depending upon a string found in the message header. As an administrator, you can create channel-level filters and MTA-wide filters to prevent delivery of unwanted mail. Users can create per-user filters for their own mailboxes by using the mail filter interface in either Communications Express or Convergence.

The server applies filters in the following priority:

1. User-level filters
   If a personal mailbox filter explicitly accepts or rejects a message, then filter processing for that message finishes. But if the recipient user had no mailbox filter, or if the user's mailbox filter did not explicitly apply to the message in question, Messaging Server next applies the channel-level filter.
2. Channel-level filter
   If the channel-level filter explicitly accepts or rejects a message, then filter processing for that message finishes. Otherwise, Messaging Server next applies the MTA-wide filter, if there is one.
3. MTA-wide filter

By default, each user has no mailbox filter. When a user uses either the Communications Express or Convergence interfaces to create one or more filters, then their filters are stored in the Directory and retrieved by the MTA during the directory synchronization process.

To Create User-level Filters

Per-user mail filters apply to messages destined for a particular user's mailbox. Per-user mail filters can only be created by using either the Communications Express or Convergence interfaces.

To Create Channel-level Filters

Channel-level filters apply to each message enqueued to a channel. A typical use for this type of filter is to block messages going through a specific channel.

`filter` Channel Keyword URL-pattern Substitution Tags (Case-insensitive)

Tag	Meaning
*	Perform group expansion.
**	Expand the attribute mailForwardingAddress. This can be a multivalued attribute resulting in several delivery addresses being produced.
$$	Substitute in the $ character
$	Force subsequent text to lower case
$^	Force subsequent text to upper case
$_	Perform no case conversion on subsequent text
$~	Substitute in the file path for the home directory associated with the local part of the address
$1S	As $S, but if no subaddress is available just insert nothing
$2S	As $S, but if no subaddress is available insert nothing and delete the preceding character
$3S	As $S, but if no subaddress is available insert nothing and ignore the following character
$A	Substitute in the address, local-part@host.domain
$D	Substitute in host.domain
$E	Insert the value of the second spare attribute, LDAP_SPARE_1
$F	Insert the name of the delivery file (mailDeliveryFileURL attribute)
$G	Insert the value of the second spare attribute, LDAP_SPARE_2
$H	Substitute in host
$I	Insert the hosted domain (part of UID to the right of the separator specified by domainUidSeparator). Fail if no hosted domain is available
$1I	As $I, but if no hosted domain is available just insert nothing
$2I	As $I, but if no hosted domain is available insert nothing and delete the preceding character
$3I	As $I, but if no hosted domain is available insert nothing and ignore the following character
$L	Substitute in local-part
$M	Insert the UID, stripped of any hosted domain
$P	Insert the method name (mailProgramDeliveryInfo attribute)
$S	Insert the subaddress associated with the current address. The subaddress is that part of the user part of the original address after the subaddress separator, usually {+}, but can be specified by the MTA option SUBADDRESS_CHAR. Fail if no subaddress is given
$U	Insert the mailbox part of the current address. This is either the whole of the address to the left of the @ sign, or that part of the left hand side of the address before the subaddress separator, {+}.

To Create a Channel-level Filter

1. Write the filter using Sieve.
2. Store the filter in a file in the following directory:
   msg-svr-base/config/*file*.filter
   The file must be world readable and owned by the MTA's uid.
3. Include the following in the channel configuration:

```
destinationfilter file:IMTA_TABLE:<file>.filter
```

4. Recompile the configuration and restart the Dispatcher.
   Note that changes to the filter file do not require a recompile or restart of the Dispatcher.
   The `destinationfilter` channel keyword enables message filtering on messages enqueued *to* the channel to which it is applied. The `sourcefilter` channel keyword enables message filtering on messages enqueued *by* (from) the channel to which it is applied. These keywords each have one required parameter which specifies the path to the corresponding channel filter file associated with the channel.
   The syntax for the `destinationfilter` channel keyword is:
   `destinationfilter` *URL-pattern*
   The syntax for the `sourcefilter` channel keyword is:
   `sourcefilter` *URL-pattern*
   where *URL-pattern* is a URL specifying the path to the filter file for the channel in question. In the following example, *channel-name* is the name of the channel.

   ```
   destinationfilter file:///usr/tmp/filters/<channel-name>.filter
   ```

   The `filter` channel keyword enables message filtering on the channels to which it is applied. The keyword has one required parameter which specifies the path to the filter files associated with each envelope recipient who receives mail via the channel.
   The syntax for the `filter` channel keyword is:
   `filter` *URL-pattern*
   URL-pattern is a URL that, after processing special substitution sequences, yields the path to the filter file for a given recipient address. *URL-pattern* can contain special substitution sequences that, when encountered, are replaced with strings derived from the recipient address, `local-part@host.domain` in question. These substitution sequences are shown in Channel Keyword URL-pattern Substitution Tags (Case-insensitive).
   The `fileinto` keyword specifies how to alter an address when a mailbox filter `fileinto` operator is applied. The following example specifies that the folder name should be inserted as a subaddress into the original address, replacing any originally present subaddress:
   `fileinto $U+$S@$D`

To Create MTA-Wide Filters

MTA-wide filters apply to all messages enqueued to the MTA. A typical use for this type of filter is to block unsolicited bulk email or other unwanted messages regardless of the messages' destinations. To create an MTA-wide filter:

1. Write the filter using Sieve
2. Store the filter in the following file:
   msg-svr-base`/config/imta.filter`
   This filter file must be world readable. It is used automatically, if it exists.
3. Recompile the configuration and restart the Dispatcher
   When using a compiled configuration, the MTA-wide filter file is incorporated into the compiled configuration.

Routing Discarded Messages Out the FILTER_DISCARD Channel

By default, messages discarded via a mailbox filter are immediately discarded (deleted) from the system. However, when users are first setting up mailbox filters (and perhaps making mistakes), or for debugging purposes, it can be useful to have the deletion operation delayed for a period.

To have mailbox filter discarded messages temporarily retained on the system for later deletion, first add

a filter_discard channel to your MTA configuration with the notices channel keyword specifying the length of time (normally number of days) to retain the messages before deleting them, as shown in the following example:

```
filter_discard notices 7
filter-discard
```

Then set the option FILTER_DISCARD=2 in the MTA option file. Messages in the filter_discard queue area should be considered to be in an extension of users' personal wastebasket folders. As such, note that warning messages are never sent for messages in the filter_discard queue area, nor are such messages returned to their senders when a bounce or return is requested. Rather, the only action taken for such messages is to eventually silently delete them, either when the final notices value expires, or if a manual bounce is requested using a utility such as imsimta return.

Prior to Messaging Server 6 2004Q2, the use of the filter_discard channel by the jettison Sieve action was controlled by the FILTER_DISCARD MTA option. This is now controlled by the option FILTER_JETTISON, which takes its default from the FILTER_DISCARD setting. FILTER_DISCARD in turn defaults to 1 (discards go to the bitbucket channel).

To Debug User-level Filters

If a user complains that a Sieve filter is not behaving as expected, there are a number of steps you can take to debug the filters. These are described here.

1. For fileinto filtering to work, check that the ims-ms channel in the imta.cnf file is marked as follows:
   `fileinto $U+$S@$D`
2. Get the user level filters from the user's LDAP entry.
   User level filters are stored in their LDAP entry under the MailSieveRuleSource attribute(s). To retrieve this with a ldapsearch command, remember they are base64 encoded so you will need to decode the output by using the -Bo switch.

   ```
   ldapsearch -D "cn=directory manager" -w password -b
   "o=alcatraz.sesta.com,o=isp" -Bo uid=test
   ```

   The imsimta test -rewrite command, described below, will also automatically decode them.

3. Verify that the use's filters are being seen by the MTA.
   Issue the command:

   ```
   imsimta test -rewrite -filter -debug user@sesta.com
   ```

   This should output the user's Sieve filters that you retrieve in the previous step. If you do not see the filters, then you need to figure out why the LDAP entry isn't returning them. If the imsimta test -rewrite output shows the filters, then you know that the user's filters are being seen by the MTA. The next step is to test the interpretation of the filters by using the imsimta test -expression command.

4. Use imsimta test -exp to debug the user's filter. The following information is required:
   a. The user's Sieve language statements from their mailSieveRuleSource attribute. See the steps above.
   b. The rfc2822 message that was supposed to trigger the filter.
   c. Description of what they expected the filter to do to the message.

5. Create a text file (example: `temp.filter`) containing the Sieve language statements based on the user's `mailSieveRuleSource: values`. Example:

```
require "fileinto";
if anyof(header :contains
["To","Cc","Bcc","Resent-to","Resent-cc", "Resent-bcc"] "commsqa"){
   fileinto "QMSG";
}
```

   Expected result: if `commsqa` is a recipient for this message, then file the message into a folder called `QMSG`.

6. Create a text file called `test.msg` that contains the contents of the rfc2822 message file supplied by user.
   You can either use a `.msg` file from the user's message store area, or create a text file called `test_rfc2822.msg` that contains the contents of the rfc2822 message file supplied by user.
7. Use the `imsimta test -exp` command:

```
imsimta test -exp -mm -block -input=temp.filter
-message=test_rfc2822.msg
```

8. Examine the output.
   The last lines of the `imsimta test -exp` command will show the result of the Sieve interpretation. They will look like this:

```
Sieve Result: []
```

   or this:

```
Sieve Result: [_action_]
```

   where *action* is the action that would be done as a result of applying the Sieve filter on this message.
   If the criteria of the filter matched, you will have some action displayed as the result. If nothing matched, then the Sieve result will be blank, and there is either a logic error in the Sieve filter or the `.msg` file doesn't contain matching information. If you get any other error, then there is a syntax error in the Sieve script file, and you need to debug it.
   For more details on the output, see imsimta test -exp Output.

9. If the filter is syntactically valid and results are correct, then the next step is to examine a `tcp_local_slave.log` debug log file.
   It may be that the message file you're testing and the one being sent aren't identical. The only way to see what's being received is to examine a `tcp_local_slave.log` file. This log will show you the exact message being sent to the MTA and how the filter is being applied to that message.
   For more information on getting a `tcp_local_slave.log` debug file, see the `slave_debug` keyword in Configuring Channel Definitions.

`imsimta test -exp` Output

The full command `imsimta test -exp` for is as follows:

```
imsimta test -exp -mm -block -input=temp.filter -message=rfc2822.msg
```

An example of the output is as follows:

Example - `imsimta test -exp` Output

```
# imsimta test -exp -mm -block -input tmp.filter -message=rfc2822.msg
Expression: if header :contains ["to"] ["pamw"]       (1)
Expression: {
Expression: redirect "usr3@sesta.com";
Expression: keep;
Expression: }
Expression:
Expression: Dump: header:2000114;0  3  1  :contains  1  "to"  1
"pamw"  if  8  ;
Dump: redirect:2000121;0  1  1  "usr3@sesta.com"  ;  keep:2000117;0  (2)
Dump: 0
Result: 0
Filter result: [ redirect "usr3@sesta.com" keep ]    (3)
```

1) The Expression: output lines show the filter being read and parsed from `tmp.filter` text file. These are not particularly useful in debugging the script.

2) The `Dump:` output lines are the result of the computer interpreting the Sieve statements. You should not see any errors and the output should seem to match your input. For example the dump shows the word `redirect`, `usr3@sesta.com` which is like the line in the filter file `redirect "usr3@sesta.com";`.

If it didn't show this matching text, then you'd be concerned, otherwise, these also are not particularly useful in debugging the script.

3) At the bottom of the output you will get the `Filter result:` statement. As stated earlier there are two possible results:

`Sieve Result: []` or this: `Sieve Result: [`*action*`]`

where action is the action taken by the Sieve script. Note that sometimes the results are expected to be empty. For example, for a discard filter, you should test that it doesn't always discard every `.msg` file you test it against. If there is some action between the brackets, for example:

```
Filter result: [fileinto "QMSG" keep]
```

This means the text in the `rfc2822.msg` file matched the filter criteria. In this particular example, the filter will file the mail into folder `QMSG` and keep a copy in the inbox. The resulting actions in this case are `fileinto` and `keep`.

When testing filters you should test various `.msg` files for both results. You should always test that messages that match your filter are filtered, and messages that you do not want to match are not filtered.

Keep in mind that if for wildcard matches, you must use the `:matches` test and not `:contains`. For example, if you wish `from=*@sesta.com` to match, you must use `:matches` or the test will fail as it will not ever satisfy the test condition.

imsimta test -exp Syntax

`imsimta test -exp` tests Sieve language statements against a specified RFC2822 message and sends the results of the filter to standard output.

The syntax is as follows:

`imsimta test -exp -mm -block -input=`*Sieve_language_scriptfile* `-message=` *rfc2822_message_file*

where,

`-block` treats the entire input as a single Sieve script. The default is to treat each line as a separate script and to evaluate it separately. The Sieve will only be evaluated once the end of file is reached.

Chapter 20. Message Store Management

Message Store Management

The message store is the component of the Messaging Server that contains the user mailboxes as well as the servers that provide IMAP, POP, and HTTP access to the mailboxes.

> **Note**
> Unless otherwise noted, the message store (specifically, the `stored` process) should be up and running while performing management and maintenance tasks described in Message Store Command-line Utilities.

The size of the message store increases as the number of users, mailboxes, and log files increase. You can control the size of the message store by specifying limits on the size of mailboxes, by specifying limits on the total number of messages allowed, and by setting aging policies for messages in the store.

Depending on the number of users your server supports, the message store might require one physical disk or multiple physical disks. There are two ways to integrate this additional disk space into your system. The easiest way is to add additional message store partitions (see Managing Message Store Partitions and Adding Storage) Likewise, if you are supporting multiple hosted domains, you might want to dedicate a server instance to a single, large domain. With this configuration, you can designate a store administrator for a particular domain. You can also expand the message store by adding more partitions.

Topics:

- Message Store Pages by Topic
- All Message Store Pages

Message Store Pages by Topic

Page: Message Store Administration
Page: Message Store Administration in Unified Configuration
Page: Message Store Architecture and Concepts
Page: Message Store Architecture and Concepts in Unified Configuration
Page: Message Store Backup and Restore
Page: Message Store Command-line Utilities
Page: Message Store Message Expiration
Page: Message Store Message Types Overview
Page: Message Store Message Types Overview in Unified Configuration
Page: Message Store Quota (Overview)
Page: Message Store Quota (Overview) in Unified Configuration
Page: Message Store Shared Folders Overview
Page: Migrating Mailboxes to a New System
Page: Monitoring the Message Store

Page: Quotas & Automatic Mail Deletion

Page: Troubleshooting the Message Store

Page: Upgrading the Message Store

All Message Store Pages

Overview and Concepts

Page: Message Store Administration

Page: Message Store Administration in Unified Configuration

Page: Message Store Architecture and Concepts

Page: Message Store Architecture and Concepts in Unified Configuration

Page: Message Store Automatic Recovery on Startup

Page: Message Store Command-line Utilities

Page: Message Store Directory Layout

Page: Message Store Load Throttling

Page: Message Store Logging

Page: Message Store Maintenance Queue

Page: Message Store Message Types Overview

Page: Message Store Message Types Overview in Unified Configuration

Page: Message Store Shared Folders Overview

Page: Message Store Valid UIDs and Folder Names

Page: Upgrading the Message Store

Page: Valid Message Store UIDs and Folder Names in Unified Configuration

Message Store Tasks and Examples

Page: Backing Up and Restoring the Message Store

Page: Backing Up and Restoring the Message Store in Unified Configuration

Page: Best Practices for Messaging Server and ZFS

Page: Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS in Unified Configuration

Page: Configuring Message Expiration

Page: Configuring Message Expiration in Unified Configuration (Tasks)

Page: Configuring POP, IMAP, and HTTP Services

Page: Managing Mailboxes

Page: Managing Message Store Partitions and Adding Storage

Page: Managing Message Store Partitions and Adding Storage in Unified Configuration

Page: Managing Message Store Quotas

Page: Managing Message Store Quotas in Unified Configuration

Page: Managing Message Types in the Message Store

Page: Managing Message Types in the Message Store in Unified Configuration
Page: Managing Shared Folders
Page: Managing Shared Folders in Unified Configuration
Page: Migrating Mailboxes to a New System
Page: Monitoring Disk Space
Page: Monitoring the Message Store
Page: Monitoring User Access to the Message Store
Page: moveuser
Page: Protecting Mailboxes from Deletion or Renaming
Page: Protecting Mailboxes from Deletion or Renaming (Unified Configuration)
Page: Reducing Message Store Size Due to Duplicate Storage
Page: Reducing Message Store Size Due to Duplicate Storage in Unified Configuration
Page: Specifying Administrator Access to the Message Store
Page: Troubleshooting the Message Store

Administering Message Store Database Snapshots (Backups)

Administering Message Store Database Snapshots (Backups)

This information describes the tasks for administering database snapshots. For conceptual information on database snapshots, see Message Store Automatic Recovery on Startup.

The primary message store database is critical to smooth operation of the message store. You must always ensure that a snapshot (or backup) of the database is available. If the active database becomes damaged, restarting services allows the message store `stored` process to swap in the best snapshot and enable services to come back on demand.

Topics:

To Specify Message Store Database Snapshot Interval and Location

Before doing these tasks, see Message Store Database Snapshot Interval and Location. Also, descriptions of the `configutil` parameters described at http://msg.wikidoc.info/index.php/Configutil_Reference.

A database snapshot is a hot backup (dynamic backup) of the message store database. The system makes this copy without any locking, and requires that both the database files and transaction logs so the snapshot can be "recovered" into a valid copy of the database.

Beginning in Messaging Server 7.0, by default, snapshots are scheduled with the `imdbverify -s` command to run at specific times.

```
local.schedule.snapshot.enable = "1"
local.schedule.snapshot = "0 2 * * * bin/imdbverify -s -m"
```

By default, the `imdbverify -s` command takes a database snapshot at 2 a.m. (This command uses UNIX crontab format: minute hour day-of-month month-of-year day-of-week command arguments.) The `-m` option is used to verify the snapshot. The `-m` option is not required. See Message Store Database Snapshot Recovery and Verification for more information.

Database snapshots are located in the following base directory:

```
local.store.snapshotpath = "dbdata/snapshots"
```

Change this directory to a different disk than the disk used by the primary database, for both performance and recovery reasons.

You configure the number of snapshots retained over time with the following parameter:

```
local.store.snapshotdirs = "3"
```

Each snapshot requires as much disk space as the entire database and transaction logs at any given time.

Take enough snapshots such that you both have a recent copy, and copies that go back a day or two, to be sure you can find a database not affected by any odd system problems that are not immediately discovered.

Prior to Messaging Server 7.0, snapshots were previously taken by the `stored` daemon, timed by the interval defined in this now obsolete configuration variable:

```
local.store.snapshotinterval
```

Message Store Database Snapshot Recovery and Verification

Beginning in Messaging Server 7.0, the message store has been enhanced to continuously recover archived log files into an up-to-date backup copy of the message store database. If the actual database becomes unusable, then the message store automatically uses this backup database. Having an up-to-date database backup provides the next level of recovery and stability for the message store.

The system automatically runs `imdbverify -m` as specified for rolling backups, and `imdbverify -s -m` as specified under snapshots.

If the verification process detects any errors, the errors are written to the `default` log. Errors in the `default` log mean not only that the snapshot failed, but they could also be pointing to a problem with the active database. (However, at this time, not all verification errors indicate a live database problem.)

Message Store Database Snapshot Rolling Backup

The message store rolling backup operates as a specially designated snapshot, where each transaction log is added to the snapshot and recovered every few minutes to provide a more up-to-date backup. For this reason, always enable snapshots, and rolling backup will be enabled by default.

Rolling backup requires the following three configuration settings, which are enabled by default:

```
local.store.rollingdbbackup = "yes"

local.schedule.snapshotverify.enable = 1

local.schedule.snapshotverify = "1,3,5,7,9,11,13,15,17,19,
                                 21,23,25,27,29,31,33,35,37,
                                 39,41,43,45,47,49,51,53,55,
                                 57,59 * * * * bin/imdbverify -m"
```

In this configuration:

- `local.store.rollingdbbackup` enables rolling backup. This means the log archive function running under the `stored` daemon copies the database transaction logs to the rolling snapshot instead of removing them every minute.
- `local.schedule.snapshotverify` is addition verifies, which are required to continually roll the log files into the snapshot.

Note
Should a rolling backup fail any of its verifies, each side of the process declares the rolling backup invalid and cleans up the logs. Rolling backup then restarts after the next snapshot is put in place. Rolling backup relies on an initial snapshot taken by the normal snapshot process.

Message Store Database Recovery

When the message store services are started, the message store process `stored` decides if the current database is damaged, and if so, replaces it with the best snapshot. The best snapshot is printed in the logs and any recovery actions are also printed in the `default` log.

Administering Very Large Mailboxes

Administering Very Large Mailboxes

In Messaging Server 5 and 6, the `store.idx` files, which contain the message index and cache records, were limited to 2 Gbytes in size. Because of this limit on the `store.idx` file, the size of an actual mailbox was limited to about 1 million messages on average, depending on the header size and the complexity of the messages.

> **Messaging Server 7.0** increased the number of messages a mailbox can contain, that is, it introduced the ability to have "very large" mailboxes.

Topics:

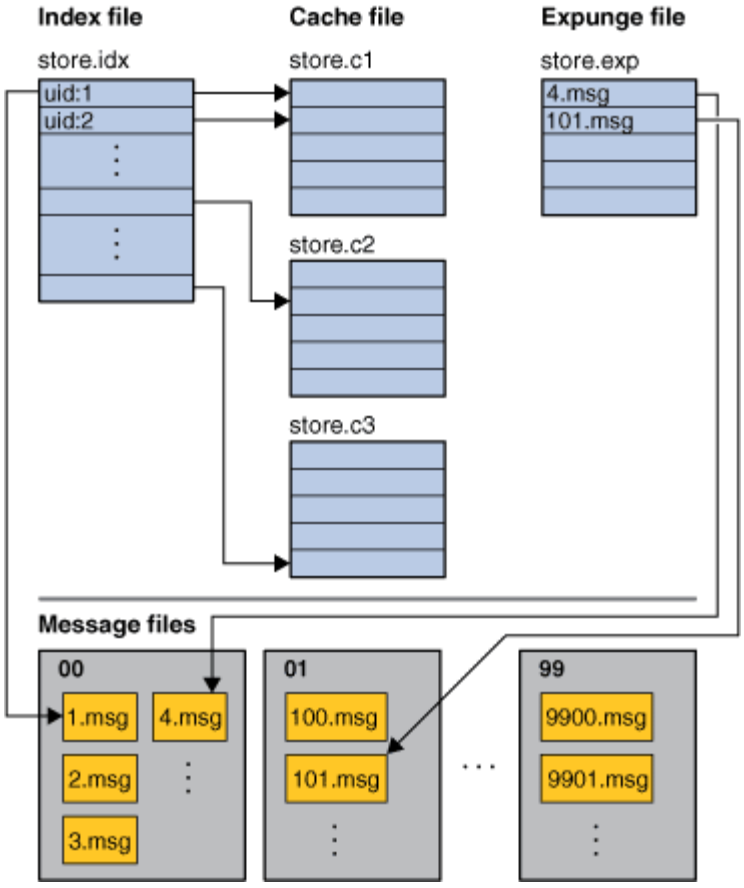
Very Large Mailboxes Overview

To increase the mailbox size limit and improve expunge performance, the index records and cache records have been split into separate files with support added for multiple cache files. An index file contains a mailbox header and a 128-byte fixed-length record for every message in the mailbox. The index record contains the location and size of the corresponding cache record. The cache files contain frequently used data in variable length records. The cache file size is configurable (`store.maxcachefilesize`). The default cache size is 500 Mbytes, with a maximum size 2 Gbytes. New cache records are appended to the newest cache file. As a cache file fills up, a new cache file is created, enabling a mailbox to continue to grow.

To optimize expunge performance, only the index file `store.idx` is purged when a mailbox is expunged. Expunge removes the obsoleted index records from the index file. Cache record and message file removal is deferred until the size of the expunged data exceeds a configurable threshold (`store.purge.count` or `store.purge.percentage` for cache records; `store.cleanupsize` for the *number* of messages). When the expunged size exceeds the threshold, expunge enqueues a purge request to the Message Store Maintenance Queue. Impurge dequeues the request, removes the unused message files and purge the cache files when the expunged size exceeds the purge threshold.

The Structure of a Mailbox

The following figure is an example of a large mailbox:



Mailbox Size Limit

IMAP defines the UID as a 32-bit value. Therefore, the maximum number of messages in a mailbox is limited to 4,294,967,295. For a 64 bit messaging server, the maximum number of messages in a folder is 4,294,967,295. For a 32-bit messaging server, the maximum number of messages in a folder is 16,777,215. The `configutil` variable `local.store.maxmessages` can be used to limit the size of a folder. The default limit is 16,000,000.

Note, the 4 billion limit is a hard limit. Effective limit is much smaller (normally less than 1 million). Mailbox expunge has to rewrite the `store.idx file`. The larger the mailbox, the longer it takes to expunge.

Mailbox Migration

Mailboxes are migrated to the new format automatically when they are opened. The operation is transparent to the end users.

Pre-Deployment Preparations

The high-level steps for preparing to use very large mailboxes include:

1. Configuring the maximum cache file size
2. Configuring the mailbox expunge size
3. Configuring `local.store.maxmessages`, quotas or expire rules to prevent mailboxes from getting too big

The new `configutil` parameters to manage large mailboxes are:

configutil Parameter	Description
`store.maxcachefilesize`	Sets the maximum cache file size in bytes, maximum of 2 Gbytes
`store.cleanupsize`	Cleans the mailbox when the number of expunged messages exceeds this value. Default is 100

Checking Mailbox Data

The `imcheck` command line utility can be used to dump the data of mailboxes in readable format. For example, `imcheck -m` *`mailbox`* dumps the content of a `store.idx` file:

```
$ imcheck -m user/dumbo/INBOX
-------------------------------------------------------------------------------
user/dumbo
Version: 103
Exists: 4
Flags: 0
Largest Msg: 1521 bytes
Last Append: 20080131104858
Last Repair:              -
Last UID: 4
Oldest Msg: 20080131104843
Oldest Uid: 1
Quota Used: 2394
Bytes Expunged: 0
UID Validity: 1201805323
Last CacheId: 1
Start Offset: 256
Append CacheId: 1
ACL: dumbo      lrswipcdan
Subscribed: 0
Partition: primary
Path: /var/opt/SUNWmsgsr/store/partition/primary/=user/94/60/=dumbo
Msg Path: /var/opt/SUNWmsgsr/store/partition/primary/=user/94/60/=dumbo

 MsgNo    Uid  Internal-Date      Sent-Date     Size HSize Cache-Id
C-Offset C-Len   Last-Updated      Save-Date MT SFlags UFlags
Original-Uid    Message-id
-------------------------------------------------------------------------------
1      1 20080131104843 20080131104843      257   244        1
16   864 20080131104843 20080131104843  2 R      0.0.0 1201805323-1
-
     2      2 20080131104851 20020301191039      338   229        1
880   816 20080131104851 20080131104851  2 R      0.0.0 1201805323-2
<001@red.iplanet.com>
     3      3 20080131104855 20020301191039     1521   241        1
1696   840 20080131104855 20080131104855  2 R      0.0.0 1201805323-3
<002@red.iplanet.com>
     4      4 20080131104858 20050919110532      278   252        1
2536   860 20080131104858 20080131104858  1 R      0.0.0 1201805323-4
<003@red.iplanet.com>
```

`imcheck -m mailbox -c msgno` displays the cache records of a message:

```
$ imcheck -m user/dumbo/INBOX -c 1
Cache items of user/dumbo message number 1:

ENVELOPE {300}
("Thu, 31 Jan 2008 10:48:43 -0800" "welcome" (("Mail Administrator" NIL
"Postmaster" "puzzle.red.iplanet.com")) (("Mail Administrator" NIL
"Postmaster" "puzzle.red.iplanet.com")) (("Mail Administrator" NIL
"Postmaster" "puzzle.red.iplanet.com")) ((NIL NIL "dumbo"
"red.iplanet.com")) NIL NIL NIL NIL)

BODYSTRUCTURE {75}
("TEXT" "PLAIN" ("CHARSET" "us-ascii") NIL NIL "7BIT" 13 1 NIL NIL NIL
NIL)

BODY {59}
("TEXT" "PLAIN" ("CHARSET" "us-ascii") NIL NIL "7BIT" 13 1)

SECTION {48}
Header offset: 0 len: 244
Body offset: 244 len: 13
1 subparts
(1) offset: 244 len: 13 charset: 0 encoding: 0

CACHEHEADERS {244}
Subject: welcome
To: dumbo@red.iplanet.com
Date: Thu, 31 Jan 2008 10:48:43 -0800
From: Mail Administrator <Postmaster@puzzle.red.iplanet.com>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit



FROM {53}
mailadministrator <postmaster@puzzle.red.iplanet.com>

TO {23}
<dumbo@red.iplanet.com>

CC {0}


BCC {0}


SUBJECT {9}
"welcome"

XSENDER {0}
```

Backing Up and Restoring the Message Store

Backing Up and Restoring the Message Store

This information describes how to back up and restore mailboxes. For conceptual information on the message store, see the following:

- Administering Message Store Database Snapshots (Backups))
- Message Store Disaster Backup and Recovery and Message Store Directory Layout)

Topics:

Mailbox Backup and Restore Overview

Mailbox backup and restore is one of the most common and important administrative tasks. You must implement a backup and restore policy for your message store to ensure that data is not lost if the following problems occur:

- System crashes
- Hardware failure
- Accidental deletion of messages or mailboxes
- Problems when reinstalling or upgrading a system
- Natural disasters (for example, earthquakes, fire, hurricanes)
- Migrating users

You can back up and restore mailboxes by using the command-line utilities `imsbackup` and `imsrestore`, or the integrated backup and restore solution that uses Oracle StorageTek Enterprise Backup Software (EBS).

Messaging Server provides a single-copy backup procedure. Regardless of how many user folders contain a particular message, during backup, the message file is backed up only once using the first message file found. The second message copy is backed up as a link to the name of the first message file, and so on. `imsbackup` maintains a hash table of all messages using the device and inode of the message files as the index. This method does have implications when restoring data, however. For more information, see Considerations for Partial Restore.

> **Note**
> You can also back up and restore the message store by backing up all relevant message files and directories. See Message Store Disaster Backup and Recovery for more information.

Backing up mailboxes includes three steps:

1. To Create a Mailbox Backup Policy

2. To Create Backup Groups
3. To Run the imsbackup Utility

To Create a Mailbox Backup Policy

Your backup policy will depend on several factors, such as:

- Peak Business Loads
- Full and Incremental Backups
- Parallel or Serial Backups

Peak Business Loads

Take into account peak business loads when scheduling backups for your system as this can reduce system load during peak hours. For example, backups are probably best scheduled for early morning hours such as 2:00 AM.

Full and Incremental Backups

Incremental backups (see Incremental Backup) scan the message store for changed data and back up only what has changed. Full backups back up the entire message store. Determine how often the system should perform full as opposed to incremental backups. For example, you probably want to perform incremental backups as a daily maintenance procedure and full backups once a week.

Parallel or Serial Backups

When user data is stored on multiple disks, you can back up user groups in parallel. Depending on system resources, parallel backups can speed up the overall backup procedure. However, you might want to use serial backups to reduce backup impact on the server's performance. Whether to use parallel or serial backups can depend on many factors, including system load, hardware configuration, how many tape drives are available, and so on.

To Create Backup Groups

A backup group is an arbitrary set of user mailboxes defined by regular expressions. By organizing user mailboxes into backup groups, you can define more flexible backup management.

For example, you could create three backup groups, the first containing user IDs starting with the letters A through L, the second with users whose user IDs begin with M through Z, and the third with users whose user IDs begin with a number. Administrators could use these backup groups to back up mailboxes in parallel, or perhaps only certain groups on one day and other groups on another.

Consider the following points about backup groups:

1. They are arbitrary *virtual* groups of mail users that do not precisely map to the message store directory, although backup groups could resemble the message store directory.
2. Administrators define backup groups by using UNIX regular expressions. The regular expressions are defined in the msg-svr-base_/`config/backup-groups.conf` file.
3. When backup groups are referenced in `imsbackup` and `imsrestore`, they use the path format: */partition_name/backup_group*
4. When you run the `imsbackup` command, it evaluates the entire `backup-groups.conf`, and if it finds more than one group that matches a user, it uses the first match.
   For example, the following `backup-groups.conf` contains these definitions:

```
groupA=a.*
...
groupN=.*n$
```

Because both groups match the user ID `admin`, the `imsbackup` command uses the first match, which is `groupA`. Thus, `groupA` includes the `admin` mailbox. Furthermore, the `groupN` backup does not include the `admin` mailbox.

The format of `backup-groups.conf` is as follows:

```
group_name=definition
group_name=definition
.
.
.
```

Using the example described in the previous paragraph, you would use the following definitions to create the three backup groups:

```
groupA=[a-l].*
groupB=[m,-z].*
groupC=[0-9].*
```

You can now scope `imsbackup` and `imsrestore` at several levels. You can backup the whole message store by using the following backup commands:

```
imsbackup -f <device> /
```

To back up all mailboxes for all users in `groupA` use the following command:

```
imsbackup -f <device> /<partition>/groupA
```

The default partition is called `primary`.

Pre-defined Backup Group

Messaging Server includes one predefined backup group that is available without creating the `backup-groups` configuration file. This group is called `user` and includes all users. For example, the following command backs up all users on the `primary` partition:

```
imsbackup -f backupfile /primary/user
```

To Run the imsbackup Utility

To back up and restore your mailboxes, Messaging Server provides the `imsbackup` and `imsrestore` utilities. The `imsbackup` and `imsrestore` utilities do not have the advanced features found in general purpose tools like StorageTek Enterprise Backup Software (EBS). For example, the utilities have only very limited support for tape auto-changers, and they cannot write a single store to multiple concurrent devices. Comprehensive backup is achieved by using plug-ins to generalized tools like EBS. For more information about using EBS, see To Use StorageTek Enterprise Backup Software.

Running the imsbackup Utility

With `imsbackup`, you can write selected contents of the message store to any serial device, including magnetic tape, a UNIX pipe, or a plain file. The backup or selected parts of the backup can later be recovered by using the `imsrestore` utility. The output of `imsbackup` can be piped to `imsrestore`.

The following example backs up the entire message store to `/dev/rmt/0`:

```
imsbackup -f /dev/rmt/0 /
```

This example backs up the mailboxes of user ID `joe` to `/dev/rmt/0`:

```
imsbackup -f /dev/rmt/0 /primary/user/joe
```

This example backs up all the mailboxes of all the users defined in the backup group `groupA` to `backupfile` (see To Create Backup Groups):

```
imsbackup -f /primary/groupA > backupfile
```

Incremental Backup

The following example backs up messages stored from May 1, 2004 at 1:10 pm to the present. The default is to back up all the messages regardless of their dates:

```
imsbackup -f /dev/rmt/0 -d 20040501:131000 /
```

This command uses the default blocking factor of 20. For a complete syntax description, see the `imsbackup` page.

Regarding date-time stamp:

```
20040501:131000
YYYYMMDD:HHMMSS

2004 05 01 : 13 10 00
YYYY MM DD : HH MM SS
```

Excluding Bulk Mail When You Perform Backups

When you perform a backup operation, you can specify mailboxes that will be excluded from being backed up. By excluding bulk or trash mailboxes that can accrue large numbers of unimportant messages, you can streamline the backup session, reduce the time to complete the operation, and minimize the disk space required to store the backup data.

To exclude mailboxes, specify a value for the `configutil` parameter `local.store.backup.exclude`.

You can specify a single mailbox or a list of mailboxes separated by the ”%’ character. (”%’ is an illegal character in a mailbox name.) For example, you could specify the following values:

```
Trash
Trash%Bulk Mail%Third Class Mail
```

In the first example, the folder `Trash` is excluded. In the second example, the folders `Trash`, `Bulk Mail`, and `Third Class Mail` are excluded.

Example commands:

```
# cd /opt/sun/comms/messaging64/bin
# ./configutil -o local.store.backup.exclude -v "Trash%Bulk Mail%Third Class
Mail"
OK SET
# ./configutil -o local.store.backup.exclude
Trash%Bulk Mail%Third Class Mail
```

The backup utility backs up all folders in a user mailbox except those folders specified in the `local.store.backup.exclude` parameter.

This feature works with the Messaging Server backup utility, StorageTek Enterprise Backup Software, and third-party backup software.

You can override the `local.store.backup.exclude` setting and back up an excluded mailbox by specifying its full logical name during the operation. For example, suppose the Trash folder has been excluded. You can still back up Trash by specifying the following:

```
/primary/user/user1/trash
```

However, if you specify

```
/primary/user/user1
```

the Trash folder is excluded.

To Restore Mailboxes and Messages

To restore messages from the backup device, use the `imsrestore` command. For example, the following command restores messages for `user1` from the file `backupfile`.

```
imsrestore -f backupfile /primary/user1
```

Considerations for Partial Restore

A partial restore is when only a part of the message store is restored. A full restore is when the entire message store is restored. The message store uses a single-copy message system. That is, only a single copy of any message is saved in the store as a single file. Any other instances of that message (for example, when a message is sent to multiple mailboxes) are stored as links to that copy. Because of this, there are implications when restoring messages. For example:

- **Full Restore.** During a full restore, linked messages still point to the same inode as the message file to which they are linked.
- **Partial Backup/Restore.** During a partial backup and partial restore, however, the single-copy characteristic of the message store might not be preserved.

The following examples demonstrate what happens to a message that is used by multiple users when a partial restore is performed. Assume there are three messages, all the same, belonging to three users A, B, and C, as follows:

```
A/INBOX/1
B/INBOX/1
C/INBOX/1
```

Example 1. In the first example, the system performs a partial backup and full restore procedure as follows:

1. Back up mailboxes for users B and C.
2. Delete mailboxes of users B and C.
3. Restore the backup data from step 1.

In this example, `B/INBOX/1` and `C/INBOX/1` are assigned a new inode number and the message data is written to a new place on the disk. Only one message is restored. The second message is a hard link to the first message.

Example 2. In this example, the system performs a full backup and a partial restore as follows:

1. Perform full backup.
2. Delete mailboxes for user A.
3. Restore mailboxes for user A.

A/INBOX/1 is assigned a new inode number.

Example 3. In this example, partial restore might require more than one attempt:

1. Perform full backup.
   `B/INBOX/1` and `C/INBOX/1` are backed up as links to `A/INBOX/1`.
2. Delete mailboxes for users A and B.
3. Restore mailboxes for user B.
   The restore utilities ask the administrator to restore `A/INBOX` first.
4. Restore mailboxes for users A and B.
5. Delete mailboxes for user A (optional).

Note
If you want to ensure that all messages are restored for a partial restore, you can run the `imsbackup` command with the `-i` option. The `-i` option backs up every message multiple times if necessary.

If the backup device is seekable (for example, a drive or tape), `imsrestore` seeks to the position containing `A/INBOX/1` and restores it as `B/INBOX/1`. If the backup device is non-seekable (for example, a UNIX pipe), `imsrestore` logs the object ID and the ID of the depending (linked) object to a file, and the administrator must invoke `imsrestore` again with the `-r` option to restore the missing message references.

To Restore Messages from a Mailbox that Has Been Incrementally Backed-up

If you are restoring messages from a mailbox that has been incrementally backed-up, and if that mailbox exists on the server on which you want to restore the messages, then restoring the messages requires a straightforward `imesrestore`. However, if you want to restore messages from a mailbox that has been incrementally backed-up, and if that mailbox no longer exists, you must follow different restore procedures.

Use one of the following procedures to restore messages to a mailbox that does not exist on the message store server:

- During the restore operation, disable delivery of messages to the user. Do this by setting the LDAP attribute `mailDeliveryOption` to `hold`.
- Before you use `imesrestore`, create the mailbox with the `mboxutil -c` command.

The reason why these instructions must be followed for restoring an incremental backup is as follows: When a mailbox has been deleted or is being migrated, the `imsrestore` utility recreates the mailbox with the mailbox unique identification validity and message unique identifications (UIDs) stored in the backup archive.

In the past, when `imsrestore` would recreate a deleted or migrated mailbox, it would assign a new UID validity to the mailbox and new UIDs to the messages. In that situation, a client with cached messages would have to resynchronize the mailbox UID validity and message UIDs. The client would have to download the new data again, increasing the workload on the server.

With the new `imsrestore` behavior, the client cache remains synchronized, and the restore process operates transparently with no negative impact on performance.

If a mailbox exists, `imsrestore` assigns new UIDs to the restored messages so that the new UIDs remain consistent with the UIDs already assigned to existing messages. To ensure UID consistency, `imsrestore` locks the mailbox during the restore operation. However, because `imsrestore` now uses the mailbox UID validity and message UIDs from the backup archive instead of assigning new UID values, UIDs could become inconsistent if you perform incremental backups and restores.

If you perform incremental backups with the `-d` date option of the `imsbackup` utility, you might have to invoke `imsrestore` multiple times to complete the restore operation. If incremental backups were performed, you must restore the latest full backup and all subsequent incremental backups.

New messages can be delivered to the mailbox between the restore operations, but in this case, the message UIDs can become inconsistent. To prevent inconsistency in the UIDs, you need to take one of the actions previously described on this page.

To Use StorageTek Enterprise Backup Software

Messaging Server includes a backup API that provides an interface with third-party backup tools, such as EBS. The physical message store structure and data format are encapsulated within the backup API. The backup API interacts directly with the message store. It presents a logical view of the message store to the backup service. The backup service uses the conceptual representation of the message store to store and retrieve the backup objects.

Messaging Server provides an Application Specific Module (ASM) that can be invoked by the EBS's `save` and `recover` commands to back up and restore the message store data. The ASM then invokes the Messaging Server `imsbackup` and `imsrestore` utilities.

> **Note**
> This section provides information about how to use EBS with the Messaging Server message store. To understand the EBS interface, see your StorageTek Enterprise Backup Software documentation.

1. Create a symbolic link from `/usr/lib/nsr/imsasm` to `<msg-svr-base>/lib/msg/imsasm`.
2. From Oracle or EMC, obtain a copy of the `nsrfile` binary and copy it to the following directory: `/usr/bin/nsr`
   This is required only if you are using an older version of Networker (5.x). With Networker 6.0 and above, `nsrfile` is automatically installed under `/usr/bin/nsr`.
3. If you want to back up users by groups, perform the following steps:
   a. Create a backup group file as described in To Create Backup Groups.
   b. To verify your configuration, run `mkbackupdir.sh`.
      Look at the directory structure created by `mkbackupdir.sh`. The structure should look similar to that shown in Message Store Directory Layout.
      If you do not specify a `backup-groups.conf` file, the backup process uses the default backup group `ALL` for all users.
4. In the directory `/nsr/res/`, create a `res` file for your save group to invoke the `mkbackupdir.sh` script before the backup. See Message Store Directory Layout for an example.

> **Note**
> Earlier versions of Networker have a limitation of 64 characters for the save set name. If the name of this directory plus the logical name of the mailbox (for example, `/primary/groupA/fred`) is greater than 64 characters, then you must run `mkbackupdir.sh -p`. Therefore, you should use a short path name for the `-p` option of `mkbackupdir.sh`. For example the following command will create the backup image under the directory `/backup`:
>
> ```
> mkbackupdir.sh -p /backup
> ```
>
> Important: The backup directory must be writable by the message store owner (example: `mailsrv`).

The following is a sample backup groups directory structure.

```
/backup/primary/groupA/amy
                       /bob
                       /carly
                /groupB/mary
                       /nancy
                       /zelda
                /groupC/123go
                       /1bill
                       /354hut
```

The following example shows a sample `res` file named `IMS.res` in the `/nsr/res` directory:

```
type: savepnpc;
precmd: "echo mkbackupdir started",
   "/usr/siroe/server5/msg-siroe/bin/mkbackupdir.sh -p /backup";
pstcmd: "echo imsbackup Completed";
timeout: "12:00 pm";
```

You are now ready to run the EBS interface as follows:

5. Create the Messaging Server save group if necessary.
   a. Run `nwadmin`.
   b. Select Customize | Group | Create.
6. Create a backup client using `savepnpc` as the backup command:
   a. Set the save set to the directory created by `mkbackupdir`.
      For a single session backup, use `/backup`.
      For parallel backups, use `/backup/`*`server`*`/`*`group`*. Be sure you have already created *group* as defined in To Create Backup Groups. You must also set the parallelism to the number of backup sessions. See To Back Up Data By Using StorageTek Enterprise Backup Software.
7. Select Group Control | Start to test your backup configuration.
   Example. Creating A Backup Client in EBS:
   To create a backup client in EBS. From `nwadmin`, select Client | Client Setup | Create

```
Name: siroe
Group: IMS
Savesets:/backup/primary/groupA
   /backup/secondary/groupB
   /backup/tertiary/groupC
         .
         .
Backup Command:savepnpc
Parallelism: 4
```

Restoring Data Using StorageTek Enterprise Backup Software

To recover data, you can use the EBS `nwrecover` interface or the `recover` command-line utility. The following example recovers user `a1`'s INBOX:

```
recover -a -f -s siroe /backup/siroe/groupA/a1/INBOX
```

The next example recovers the entire message store:

```
recover -a -f -s siroe /backup/siroe
```

To Use a Third Party Backup Software (Besides StorageTek Enterprise Backup Software)

Messaging Server provides two message store backup solutions, the command line `imsbackup` and the StorageTek Enterprise Backup Software. A large message store running a single `imbackup` to back up

the entire message store can take a significant amount of time. The EBS solution supports concurrent backup sessions on multiple backup devices. Concurrent backup can shorten backup time dramatically (backups of 25GB of data per hour have been achieved).

If you are using another third party concurrent backup software (for example, Netbackup), you can use the following method to integrate your backup software with the Messaging Server.

1. Divide your users into groups (see To Create Backup Groups) and create a `backup-groups.conf` file under the directory *msg-svr-base*`/config/`.

   > **Note**
   > This backup solution requires additional disk space. To backup all the groups concurrently, the disk space requirement is two times the message store size. If you do not have that much disk space, divide your users into smaller groups, and then backup a set of groups at a time. For example group1 - group5, group6 - group10. Remove the group data files after backup.

2. Run `imsbackup` to back up each group into files under a staging area.
   The command is `imsbackup -f <device>/<instance>/<group>`
   You can run multiple `imsbackup` processes simultaneously. For example:

   ```
   # imsbackup -f- /primary/groupA > /bkdata/groupA &
   # imsbackup -f- /primary/groupB > /bkdata/groupB &
   . . .
   ```

3. Use your third party backup software to back up the group data files in the staging area (in our example that is `/bkdata`).
4. To restore a user, identify the group filename of the user, restore that file from tape, and then use `imsrestore` to restore the user from the data file.

Troubleshooting Backup and Restore Problems

This section describes common backup and restore problems and their solutions.

- **Problem**: `msprobe` restarts everything during a long `imsrestore` during message store migration. This can also happen with `imsbackup`, `imsimport`, or any processing intensive utility.
- **Solution**: When `imsrestore` or any processing intensive operation takes significantly more system resources than normal, and continues doing so longer than the `msprobe` interval, there might be a temporary backlog of DB transaction log files to be cleared. If there are more files than specified in `local.store.maxlog`, then `msprobe` may erroneously restart all the processes during a restore. To prevent this from happening, disable `msprobe` during the `imsrestore`.

- **Problem**: When I do an restore of a folder or INBOX using `imsrestore` or `imsasm`, it appends all the messages in that folder onto the current folder. This results in multiple copies of the messages in that folder.
- **Solution**: Make sure the `-i` flag of `imsrestore` is not set in the `imsasm` script.

- **Problem**: I want to do an incremental backup of just new messages added in a mail folder, but when I try, the entire folder gets backed up. How do I just back up the new messages?
- **Solution**: Set the `-d` *datetime* flag on `imsbackup`. This will backup messages stored from the specified date and time to the present. The default is to back up all messages regardless of their dates.

Message Store Disaster Backup and Recovery

A disaster refers to a catastrophic failure of the entire message store as opposed to a mailbox or set of mailboxes. That is, a situation where all data on the message store servers are lost. A complete message store disaster restore will consist of restoring the following lost data:

- All message store data. These can be backed up using the procedures described in Backing Up and Restoring the Message Store. If file system backup method is used, be sure to back up the following data:
  - All message store partitions
  - The message store database files at *msg-svr-base*`/data/store/mboxlist`.
  - The message store database snapshots at *msg-svr-base* `/data/store/dbdata/snapshots` (Note that the location of message store database snapshot files can be configured with the configutil parameter `local.store.snapshotpath`.)
- Configuration data. Including the local configuration file at *msg-svr-base*`/data/config`. See also Message Store Directory Layout.

If you want to back up your message store for disaster recovery, you can use file system snapshot tools to take a snapshot of the file system. The snapshot **must** be a *point-in-time* file system snapshot.

It is best to capture all the data (message store partitions, database files and so on) at the same point-in-time, however, if this cannot be done, then you must backup the data in this order:

1. Database snapshots
2. Database files
3. Message store partitions
4. Configuration data

If the message store partitions and the database files are not backed up with the same point-in-time snapshot, run `reconstruct -m` after restoring the file system snapshots. This will synchronize the database and the store partitions.

Message Store Backup and Restore

Oracle Communications Messaging Server Message Store Backup and Restore

Best Practices for Messaging Server and ZFS

Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS

ZFS provides the following features that make it ideal for backing up the Messaging Server message store:

- Snapshot backup
- Enables the use of less expensive SATA drives
- Built-in volume manager that enables you to grow file systems dynamically

Topics:

Before You Begin

Before using ZFS to back up the Messaging Server message store, read Messaging Server and Tiered Storage Overview, which describes message store operation, its performance characteristics, and how to plan for and allocate store partitions.

Configuration Recommendations for ZFS and Messaging Server

1. Separate the Messaging Server `mboxlist` database, message file, and index cache files on different file systems.

   The `mboxlist` database is a sleepycat database that contains mailbox meta data. The index cache records (`store.idx` and `store.c*`) contain meta information about mailboxes and messages. Messaging Server accesses and modifies this meta information, although the modifications tend to be more random and smaller.

   The location of the index cache records is controlled by the `configutil` parameter `store.partition.`*partition_name*`.path`, where *partition_name* is the name of the partition.

   Each message file (`*.msg`) represents a single email. Each message file is written to disk once, never modified, read many times (for example, when a user accesses the email, when the messages are backed up, and so on) and may be deleted. By default, these files are stored with the index and cache files.

   The location of message files is controlled by the `configutil` parameter `store.partition.`*partition_name*`.messagepath`, where *partition_name* is the name of the partition.

   Separating the message files from the index cache records to different partitions (and underlying file systems) enables you to configure the file system with properties appropriate for the access type.

   Examples:

   ```
   # zfs create store/mboxlist
   # zfs set mountpoint=/var/opt/sun/comms/messaging/store/mboxlist
   store/mboxlist
   ```

```
# zfs create store/primary-idx
# configutil -o store.partition.primary.path -v /store/primary-idx
# zfs create store/primary-msg
# configutil -o store.partition.primary.messagepath -v
/store/primary-msg
```

2. Configure the index cache record file system to use the recordsize of 8 Kbytes.

   Starting with Messaging Server 7, the index recordsize is 128 bytes. (Messaging Server 5 and 6 used a smaller index record size). Cache recordsize is usually less than 2 Kbytes. The `mboxlist` database maximum page size is 8 Kbytes. The default ZFS recordsize is 128 Kbytes. Reducing the recordsize for these file systems can improve performance and reduce incremental snapshot backup size.

   Configure the index cache record file system to use 8 Kbytes recordsize and the `mboxlist` database file system to use 8 Kbytes recordsize.

   Examples:

   ```
   zfs set recordsize=8k store/primary-idx
   zfs set recordsize=128k store/primary-msg
   zfs set recordsize=8k store/mboxlist
   ```

   Note
   This setting controls the recordsize of newly created files only.

   Note
   Do not set recordsize smaller than the system page size (8 Kbytes on SPARC and 4 Kbytes on Intel).

   Note
   Set `recordsize=128k` on the `-msg` file system, even though that is the default, so that it does not accidentally get overridden by a setting on a parent file system at a later time.

   The default recordsize of 128k is appropriate for the message store message file system.

3. Disable file access time record.

   The message store does not utilize the file access time. By disabling file access time updates, you reduce unnecessary overhead.

   Examples:

   ```
   # zfs set atime=off store/mboxlist
   # zfs set atime=off store/primary-idx
   # zfs set atime=off store/primary-msg
   ```

4. Keep ZFS pool space under 80 percent utilization to maintain pool performance.

ZFS pool performance can degrade when a pool is very full. As the pool approaches 100 percent full, more time is needed to find free space and it is more likely that the free space is available only in small chunks.

Configure the disk usage alarm threshold `configutil` parameter `alarm.diskavail.msgalarmthreshold` to at least 20, to receive a warning before the disk becomes full. (The default value is 10).

Example:
```
# configutil -o alarm.diskavail.msgalarmthreshold -v 20
```

To enable message throttling sooner, configure the `configutil` parameter `local.store.diskusagethreshold` to 90.
(The default is 99).

Example:
```
# configutil -o local.store.diskusagethreshold -v 90
```

ZFS Administration Recommendations

- Perform snapshot backup regularly.
  Back up the mboxlist database, index, and message file systems atomically by using the `zfs snapshot -r` command. Then use the `zfs send` and `receive` commands, or an enterprise-level backup solution to save the data.

  Examples:

  ```
  # zfs snapshot -r store@now
  # zfs send store/mboxlist@now | ssh host2 zfs recv store/mboxlist
  # zfs send store/primary-idx@now | ssh host2 zfs recv store/primary-idx
  # zfs send store/primary-msg@now | ssh host2 zfs recv store/primary-msg
  ```

  You can use `zfs send -i` to perform incremental backups. Destroy the snapshots when they are not not needed.

  ```
  # zfs destroy -r store@now
  ```

  ZFS snapshots are for backing up and restoring the entire message store files system. You cannot back up and restore individual mailboxes. However, you can use `imsbackup` to back up the snapshot and `imsrestore` to restore the mailboxes.

Reference

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide
http://www.solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide

Messaging Server and Tiered Storage Overview

Oracle Communications Messaging Server and Tiered Storage Overview

This document describes the operation of the Oracle Communications Messaging Server message store, its performance characteristics, and how to plan for and allocate store partitions. Additionally, this document describes next generation best practices to meet the storage needs of both ISPs and enterprises.

This document contains the following sections:

Overview of Messaging Server Storage

For traditional ISPs that provide web-based email services, the rules of engagement have changed. Thanks to companies such as Google, which can now offer consumers multiple gigabytes of email storage space with unlimited retention, the threat is clear: ISPs, with their much smaller storage allotments (50-100 Mbytes) and automatic purging of messages older than 90 days, need to stay competitive by providing storage and retention capabilities similar to Google, or lose out.

The rules have also changed for enterprises and the corporate messaging market, which are being forced to comply with new regulations requiring email retention for ever longer periods of time. In fact, some companies are faced with the requirement of saving every incoming and outgoing email message forever. The storage requirements for such scenarios can be staggering, to say the least.

As if there weren't enough problems already for ISPs and enterprises, email by its nature is a very I/O intensive application where transaction speed is critical to customer satisfaction. In increasing storage capacity, businesses must ensure that email services maintain acceptable performance levels and do so in a cost-effective manner. To stay competitive, businesses understand that they must purchase large numbers of disk drives to satisfy this new appetite for storage space. Purchasing fiber channel drives is typically cost prohibitive, so ISPs and enterprises will be looking to less expensive alternatives, such as Serial Advanced Technology Attachment (SATA) drives. The challenge with SATA is that in order to reduce cost and provide higher capacity, performance is sacrificed. And there's the dilemma: both ISPs and enterprises need to dramatically increase their email storage capacity while at the same time maintaining acceptable performance levels without "breaking the bank." Customers are under immense legal and financial pressure to find a solution to this problem. The good news is that Oracle can deliver an excellent storage solution for Messaging Server deployments in the form of the Sun StorageTek 6540 Array.

The Messaging Server message store is one of the highest IOPs applications that exists. In the past, customers have kept all portions of the message store on their highest performing, most expensive disks. Fortunately, you can distributed the message store components onto different performing disks to create a cost-effective but high performing application.

Care must be taken to keep the highest IOP portion of the message store (the database and its store partition indexes) on high performance disks. The message store can generate up to 15+ IOPS per message delivered, typically many small, random writes, and is extremely sensitive to response times. If response time diminishes, it can have a cascading effect through the application. Because of the high IOP needs, the message store is ideal for Oracle's StorageTek 6540 controller.

Message Store and ZFS

Oracle's Communications Suite Deployment Engineering group has performed extensive testing with ZFS and measured its impact on the message store. The Messaging Group believes that in the future most Messaging Server customers will be using the ZFS file system. ZFS changes the workload characteristics on the file system, so that there are fewer I/Os, but the I/Os are bigger. ZFS enables read rates to diminish somewhat, whereas it enables write rates to diminish much more. In addition, ZFS can perform snapshotting and compression, which enhances the ability to back up the application. ZFS is also now supported by Oracle Solaris Cluster software.

How the Message Store Works

The Message Store is a dedicated data store for the delivery, retrieval, and manipulation of Internet mail messages. The Message Store works with the IMAP4 and POP3 client access servers to provide flexible and easy access to messaging. The Message Store also works through the HTTP server (mshttpd) to provide messaging capabilities to

Convergence and Communications Express clients in a web browser. The Message Store is organized as a set of folders or user mailboxes. The folder or mailbox is a container for messages. Each user has an INBOX where new mail arrives.

Each IMAP or Webmail user can also have one or more folders where mail can be stored. Folders can contain other folders arranged in a hierarchical tree. Mailboxes owned by an individual user are private folders. Private folders can be shared at the owner's discretion with other users on the same Message Store. Messaging Server supports sharing folders across multiple stores by using the IMAP protocol. There are two general areas in the Message Store, one for user files and another for system files. In the user area, the location of each user's INBOX is determined by using a two-level hashing algorithm. Each user mailbox or folder is represented by another directory in its parent folder. Each message is stored as a file. When there are many messages in a folder, the system creates hash directories for that folder. Using hash directories eases the burden on the underlying file system when there are many messages in a folder. In addition to the messages themselves, the Message Store maintains an index and cache of message header information and other frequently used data to enable clients to rapidly retrieve mailbox information and do common searches without the need to access the individual message files.

A Message Store can contain many message store partitions for user files. A Message Store partition is contained by a file system volume. As the file system becomes full, you can create additional file system volumes and Message Store partitions on those file system volumes to store new users.

Note
If a message store partition fills up, users on the partition are not able to store additional messages. Address this problem by using one or more of the following approaches:

- Reducing the size of user mailboxes
- If you are using volume management software, adding additional disks
- Creating additional partitions and moving mailboxes to the new partitions

The message store maintains only one copy of each message per partition. This is sometimes referred to as a single-copy message store. When the message store receives a message addressed to multiple users or a group or distribution list, it adds a reference to the message in each user's INBOX. Rather than saving a copy of the message in each user's INBOX, the message store avoids the storage of duplicate data. The individual message status flag (seen, read, answered, deleted, and so on) is maintained per folder for each user.

The system area contains information on the entire message store in a database format for faster access and no loss of service. The information in the system area can be reconstructed from the user area. Messaging Server contains a database snapshot function. When needed, you can quickly recover the database to a known state.

Messaging Server also has fast recovery, so that in case of database corruption, you can shut down the

message store and bring it back immediately without having to wait for a lengthy database reconstruction.

Messaging Server Disk Throughput

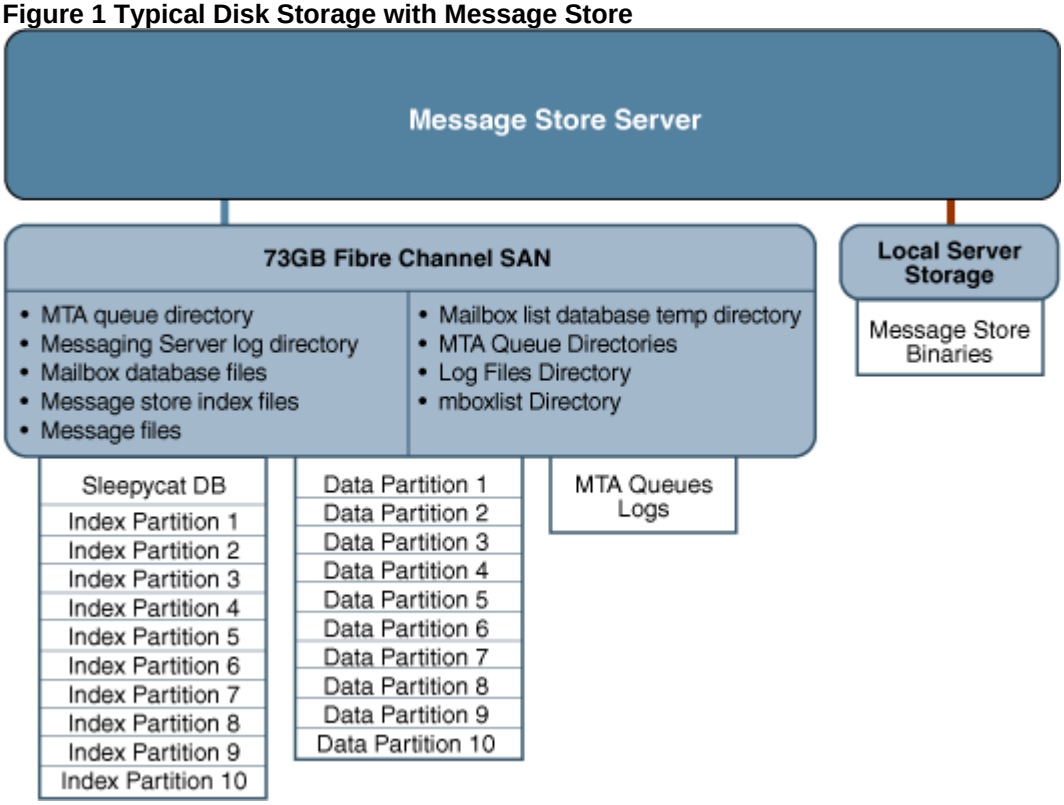
Disk throughput is the amount of data that your system can transfer from memory to disk and from disk to memory. The rate at which this data can be transferred is critical to the performance of Messaging Server. To create efficiencies in your system's disk throughput:

- Consider your maintenance operations, and ensure you have enough bandwidth for backup. Backup can also affect network bandwidth particularly with remote backups. Private backup networks might be a more efficient alternative.
- Carefully partition the store and separate store data items (such as tmp and db) to improve throughput efficiency.
- Ensure the user base is distributed across RAID (Redundant Array of Independent Disks) environments in large deployments
- Stripe data across multiple disk spindles in order to speed up operations that retrieve data from disk.
- Allocate enough CPU resources for RAID support, if RAID does not exist on your hardware.

> **Important**
> Measure disk I/O in terms of IOPS (total I/O operations per second) not bandwidth. You need to measure the number of unique disk transactions the system can handle with a very low response time (less than 10 milliseconds).

Typically, most customers deploy their entire message store on the highest performing disk as shown in the following figure.

Figure 1 Typical Disk Storage with Message Store



Messaging Server Disk Capacity

When planning server system disk space, be sure to include space for operating environment software,

Messaging Server software, and message content and tracking. Be sure to use an external disk array if availability is a requirement. For most systems, external disks are required for performance because the internal system disks supply no more than four spindles. For the Message Store partitions, the storage requirement is the total size of all messages plus 30 percent overhead. In addition, user disk space needs to be allocated. Typically, this space is determined by your site's policy.

Disk Sizing for MTA Message Queues

The purpose of the Messaging Server MTA Queue is to provide a transient store for messages waiting to be delivered. Messages are written to disk in a persistent manner to maintain guaranteed service delivery. If the MTA is unable to deliver the message, it retries delivery. At some point, if delivery is still unsuccessful, the MTA no longer tries to send the message and returns it to the sender.

MTA Message Queue Performance

Sizing the MTA Message Queue disks is an important step for improving MTA performance. The MTA's performance is directly tied to disk I/O above any other system resource. This means that you should plan on a disk volume that consists of multiple disk spindles which are concatenated and striped by using a disk RAID system. End users are quickly affected by the MTA performance. As users press the SEND button on their email client, the MTA does not fully accept receipt of the message until the message has been committed to the MTA Message Queue. Therefore, improved performance on the MTA Message Queue results in better response times for the end-user experience.

MTA Message Queue Availability

SMTP services are considered a guaranteed message delivery service. This is an assurance to end users that the messaging server will not lose messages that the service is attempting to deliver. When you architect the design of the MTA Message Queue system, all effort should be made to ensure that messages will not be lost. This guarantee is usually made by implementing redundant disk systems through various RAID technologies.

MTA Message Queue Available Disk Sizing

The MTA Message Queue grows excessively if one of the following conditions occurs:

- The site has excessive network connectivity issues
- The MTA configuration is holding on to messages too long
- Valid problems are occurring with those messages (not covered in this document)
- Message stores are operationally down, for example, for maintenance
- Remote target sites are unavailable or overwhelmed

Performance Considerations for a Message Store Architecture

Message store performance is affected by a variety of factors, including:

- Disk I/O
- Inbound message rate (also known as message insertion rate)
- Message sizes
- Login rate (POP/IMAP/HTTP)
- Transaction rate for IMAP and HTTP
- Concurrent number of connections for the various protocols
- Network I/O
- Use of SSL

The preceding factors list the approximate order of impact to the Message Store. Most performance issues with the Message Storage arise from insufficient disk I/O capacity. Additionally, the way in which you lay out the store on the physical disks can also have a performance impact. For smaller standalone systems, it is possible to use a simple stripe of disks to provide sufficient I/O. For most larger systems, segregate the file system and provide I/O to the various parts of store.

Note
In a deployment using LMTP, the MTA queue is almost always unused.

Messaging Server Directories (General Recommendations for Storage)

Messaging Server uses six directories that receive a significant amount of input and output activity. Because these directories are accessed very frequently, you can increase performance by providing each of those directories with its own disk, or even better, providing each directory with a Redundant Array of Independent Disks (RAID).

The six directories are:

- MTA Queue Directory
- Messaging Server Log Directory
- Mailbox Database Files
- Message Store Index Files
- Message Files
- Mailbox List Database Temporary Directory

MTA Queue Directory

Recommendation: Can be located on slower disks or on shared NAS
In this directory, many files are created, one for each message that passes through the MTA channels. After the file is sent to the next destination, the file is then deleted. The directory location can be changed by making the queue directory a symlink.
Default location of the MTA Queue directory: `/var/opt/sun/comms/messaging/queue`

Messaging Server Log Directory

Recommendation: Can be located on slower disks
This directory contains log files which are constantly being appended with new logging information. The number of changes depend on the logging level set. The directory location is controlled by the `configutil` parameter `logfile.*.logdir`, where `*` can be a log-generating component such as admin, default, http, imap, or pop. The MTA log files can be changed by making that directory a symlink.
Default location of the Messaging Server log directory: `/var/opt/sun/comms/messaging/log`

Mailbox Database Files

Recommendation: Keep on a fast disk
These files require constant updates as well as cache synchronization. Put this directory on your fastest disk volume. These files are always located in the `/var/opt/sun/comms/messaging/store/mboxlist` directory.

Message Store Index Files

Recommendation: Keep on fast disk
These files contain meta information about mailboxes, messages, and users. By default, these files are stored with the message files. The `configutil` parameter `store.partition.*.path`, where `*` is the name of the partition, controls the directory location. If you have the resources, put these files on your second fastest disk volume.
Default location of the message store index file:
`/var/opt/sun/comms/messaging/store/partition/primary`

Message Files

Recommendation: Can be located on slower disks

These files contain the messages, one file per message. Files are frequently created, never modified, and eventually deleted. By default, they are stored in the same directory as the message store index files. The location can be controlled with the configutil parameter `store.partition.`*`partition_name`*`.messagepath`, where
partition_name is the name of the partition. Some deployments might use a single message store partition called **primary** specified by the `store.partition.primary.path`. Large sites might have additional partitions that can be specified with `store.partition.`*`partition_name`*`.messagepath`, where *partition_name* is the name of the partition.
Default location of the message files:
`/var/opt/sun/comms/messaging/store/partition/primary`

> **Note**
> To separate the message files from the index files (enabling tiered storage), the `store.partition.*` and `.messagepath` parameters are key. These parameters must be correctly configured to put the message files on a SATA file system when you create the partition.

Mailbox List Database Temporary Directory

Recommendation: Can be located on fast disk
This is the directory used by the message store for all temporary files. To maximize performance, this directory should be located on the fastest file system. For Solaris OS, use the `configutil` command to configure the `store.dbtmpdir` variable to a directory under `tmpfs`, for example, `/tmp/mboxlist`.
Default location of the mailbox list database: `/var/opt/sun/comms/messaging/store/mboxlist`

The following sections provide more detail on Messaging Server high-access directories.

Additional Information: MTA Queue Directories

In non-LMTP environments, the MTA queue directories in the Message Store system are also heavily used. LMTP works such that inbound messages are not put in MTA queues but directly inserted into the store. This message insertion lessens the overall I/O requirements of the message store machines and greatly reduces use of the MTA queue directory on Message Store machines. If the system is standalone or uses the local MTA for Webmail sends, significant I/O can still result on this directory for outbound mail traffic. In a two-tiered environment using LMTP, this directory is lightly used, if at all. In prior releases of Messaging Server, on large systems this directory set needs to be on its own stripe or volume.

MTA queue directories should usually be on their own file systems, separate from the message files in the message store. The message store has a mechanism to stop delivery and appending of messages if the disk space drops below a defined threshold. However, if both the log and queue directories are on the same file system and keep growing, you will run out of disk space and the message store will stop working.

Additional Information: Log Files Directory

The log files directory requires varying amounts of I/O depending on the level of logging that is enabled. The I/O on the logging directory, unlike all of the other high I/O requirements of the Message Store, is asynchronous. For typical deployment scenarios, do not dedicate an entire LUN for logging. For very large store deployments, or environments where significant logging is required, a dedicated LUN is in order.

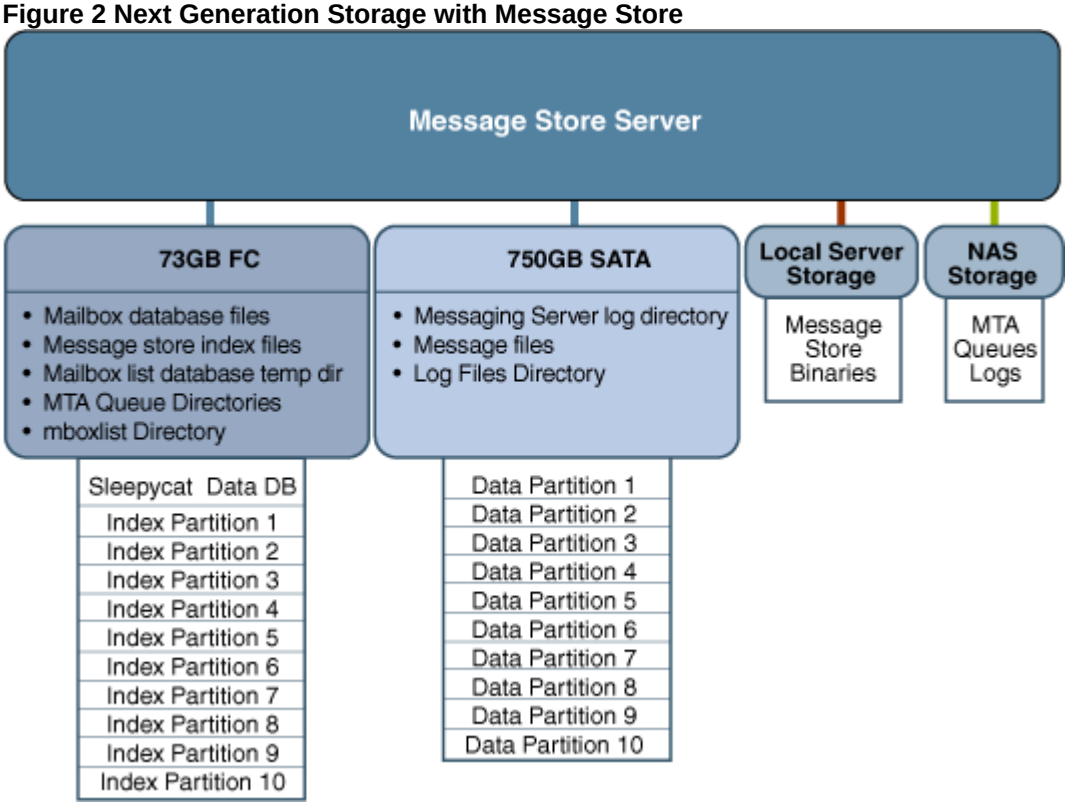
Note
In almost all environments, you need to protect the message store from loss of data. The level of loss and continuous availability that is necessary varies from simple disk protection such as RAID5, to mirroring, to routine backup, to real time replication of data, to a remote data center. Data protection also varies from the need for Automatic System Recovery (ASR) capable machines, to local HA capabilities, to automated remote site failover. These decisions impact the amount of hardware and support staff required to provide service.

Additional Information: mboxlist Directory

The `mboxlist` directory is highly I/O intensive but not very large. The `mboxlist` directory contains the databases that are used by the stores and their transaction logs. Because of its high I/O activity, and due to the fact that the multiple files that constitute the database cannot be split between different file systems, you should place the `mboxlist`directory on its own stripe or volume in large deployments. This is also the most likely cause of a loss of vertical scalability, as many procedures of the message store access the databases. For highly active systems, this can be a bottleneck. Bottlenecks in the I/O performance of the mboxlist directory decrease not only the raw performance and response time of the store but also impact the vertical scalability.

For systems with a requirement for fast recovery from backup, place this directory on Solid State Disks (SSD) or a high performance caching array to accept the high write rate that an ongoing restore with a live service will place on the file system. The following figure depicts this configuration.

Figure 2 Next Generation Storage with Message Store



Multiple Store Partitions

The Message Store supports multiple store partitions. Place each partition on its own stripe or volume. The number of partitions that should be put on a store is determined by a number of factors. The obvious factor is the I/O requirements of the peak load on the server. By adding additional file systems as additional store partitions, you increase the available IOPS (total IOs per second) to the server for mail delivery and retrieval. In most environments, you get more IOPS out of a larger number of smaller stripes

or LUNs than a small number of larger stripes or LUNs. With some disk arrays, it is possible to configure a set of arrays in two different ways. You can configure each array as a LUN and mount it as a file system. Or, you can configure each array as a LUN and stripe them on the server. Both are valid configurations.

However, multiple store partitions (one per small array or a number of partitions on a large array striping sets of LUNs into server volumes) are easier to optimize and administer. Raw performance, however, is usually not the overriding factor in deciding how many store partitions you want or need. In corporate environments, it is likely that you need more space than IOPS. Again, it is possible to software stripe across LUNs and provide a single large store partition. However, multiple smaller partitions are generally easier to manage. The overriding factor of determining the appropriate number of store partitions is usually recovery time.

Recovery times for store partitions fall into a number of categories:

- First of all, the `fsck` command can operate on multiple file systems in parallel on a crash recovery caused by power, hardware, or operating system failure. If you are using a journaling file system (highly recommended and required for any HA platform), this factor is small.

- Secondly, backup and recovery procedures can be run in parallel across multiple store partitions. This parallelization is limited by the vertical scalability of the mboxlist directory as the Message Store uses a single set of databases for all of the store partitions. Store cleanup procedures (`expire` and `purge`) run in parallel with one thread of execution per store partition.

- Lastly, mirror or RAID re-sync procedures are faster with smaller LUNs. There are no hard and fast rules here, but the general recommendation in most cases is that a store partition should not encompass more than 10 spindles. The size of drive to use in a storage array is a question of the IOPS requirements versus the space requirements. For most residential ISP POP environments, use "smaller drives." Corporate deployments with large quotas should use "larger" drives. Again, every deployment is different and needs to examine its own set of requirements.

Setting Disk Stripe Width

When setting disk striping, the stripe width should be about the same size as the average message passing through your system. A stripe width of 128 blocks is usually too large and has a negative performance impact. Instead, use values of 8, 16, or 32 blocks (4, 8, or 16 kilobyte message respectively).

MTA Performance Considerations

MTA performance is affected by a number of factors including, but not limited to:

- Disk performance
- Use of SSL
- The number of messages/connections inbound and outbound
- The size of messages
- The number of target destinations/messages
- The speed and latency of connections to and from the MTA
- The need to do spam or virus filtering
- The use of Sieve rules and the need to do other message parsing (like use of the conversion channel)

The MTA is both CPU and I/O intensive. The MTA reads from and writes to two different directories: the queue directory and the logging directory. For a small host (four processors or fewer) functioning as an MTA, you do not need to separate these directories on different file systems. The queue directory is written to synchronously with fairly large writes. The logging directory is a series of smaller asynchronous and sequential writes. On systems that experience high traffic, consider separating these two directories onto two different file systems. In most cases, you will want to plan for redundancy in the MTA in the disk

subsystem to avoid permanent loss of mail in the event of a spindle failure. (A spindle failure is by far the single most likely hardware failure.) This implies that either an external disk array or a system with many internal spindles is optimal.

MTA and RAID Trade-offs

There are trade-offs between using external hardware RAID controller devices and using JBOD arrays with software mirroring. The JBOD approach is sometimes less expensive in terms of hardware purchase but always requires more rack space and power. The JBOD approach also marginally decreases server performance, because of the cost of doing the mirroring in software, and usually implies a higher maintenance cost. Software RAID5 has such an impact on performance that it is not a viable alternative. For these reasons, use RAID5 caching controller arrays if RAID5 is preferred.

Background: Communication Services Logical Architectures Overview

You can deploy Communications Services in either a single-tiered or two-tiered logical architecture. Deciding on your logical architecture is crucial, as it determines which machine types you will need, as well as how many. In general, enterprise corporate deployments use a single-tiered architecture while internet service providers (ISPs) and telecommunications deployments use a two-tiered architecture. However, as with all generalities, the exceptions prove the rule. Small ISPs might just as well deploy on a single machine, and larger, centralized enterprises might deploy in a two-tiered architecture for many of the same reasons
that ISPs do. As more and more corporations look to offer ease of access to employees working remotely, their deployments will begin to look more and more like an ISP.

Two-tiered Logical Architecture

In a two-tiered logical architecture, the data stores communicate through front-end processes. In the case of Messaging Server, this means MMPs, MEMs, and MTAs are residing on separate machines from the data store processes. A two-tiered architecture enables the mail store to offload important and common tasks and focus on receiving and delivering mail. In the case of Calendar Server, this means the HTTP service and Administration service reside on a separate machine from the store processes. In the case of Instant Messaging, this means the proxy service is residing on a separate machine from the back-end processes.

There might be some level of cohabitation with other services. For example, you could have the Calendar store and the Message Store on the same machine. Similarly, you could have the calendar front end on the MMP machine. In a two-tiered logical architecture, Directory Server is usually a complex deployment in its own right, with multi-master and replication to a set of load-balanced consumer directories.

Benefits of a Two-tiered Architecture

All services within the Communications Services offering rely on network capabilities. A two-tiered architecture provides for a network design with two separate networks: the public (user-facing) network, and the private (data center) network. Separating your network into two tiers provides the following benefits:

- **Hides Internal Networks.** By separating the public (user-facing) network and the private (data center) network, you provide security by hiding the data center information. This information includes network information, such as IP addresses and host names, as well as user data, such as mailboxes and calendar information.
- **Provides Redundancy of Network Services.** By provisioning service access across multiple front-end machines, you create redundancy for the system. By adding redundant messaging front-end servers, you improve service uptime by balancing SMTP requests to the available messaging front-end hosts.
- **Limits Available Data on Access Layer Hosts.** Should the access layer hosts be compromised, the attackers cannot get to critical data from the access hosts.
- **Offloads Tasks to the Access Layer.** By enabling the access layer to take complete ownership of a number of tasks, the number of user mailboxes on a message store increases. This is useful

because the costs of both purchase and maintenance are much higher for store servers than for access layer machines (the second tier). Access layer machines are usually smaller, do not require large amounts of disk (see MTA Performance Considerations, and are rarely backed up. A partial list of features that are offloaded by the second tier includes:

- Denial of Service protection
- SSL
- Reverse DNS
- UBE (spam) and virus scanning
- Initial authentication - Authentications to the Message Store should always succeed and the directory servers are more likely to have cached the entry recently.
- LMTP - With support for LMTP between the MTA relays and the message stores, SMTP processing is offloaded and the need to do an additional write of the message into the MTA queues on the message stores is eliminated.

- **Simplifies End-user Settings in Client Applications.** By using a two-tiered architecture, end users do not have to remember the physical name of hosts that their messaging and calendar applications connect to. The Access-Layer Application hosts provide proxies to connect end users to their assigned messaging or calendar data center host. Services such as IMAP are connected to the back-end service using LDAP information to identify the name of the user's mailbox host. For calendar services, the calendar front-end hosts provide a calendar lookup using the directory server to create a back-end connection to the user's assigned calendar store host.

  This capability enables all end users to share the same host names for their client settings. For example, instead of remembering that their message store is host-a, the user simply uses the setting of mail. The MMP provides the proxy service to the user's assigned message store. You need to provide the DNS and load balancing settings to point all incoming connections for mail to one (or more) MMPs.

  By placing Calendar Server into two tiers, more than one Calendar Server back-end server can be used. Calendar Server's group scheduling engine enables users to schedule appointments with users whose calendars are on any of the back-end Calendar Server hosts.

  An additional benefit of this proxy capability provides geographically dispersed users to leverage the same client application settings regardless of their physical location. Should a user from Europe visit California, the user will be able to connect to the immediate access server in California. The user's LDAP information will tell the access server to create a separate connection on the user's behalf to the user's message store located in Europe. Lastly, this enables you to run a large environment without having to configure user browsers differently, simplifying user support. You can move a user's mailbox from one mail store to another without contacting the user or changing the desktop.
- **Reduces Network HTTP Traffic on the Data Center.** Both the Messaging Express Multiplexor (MEM) and the Calendar Server front-end greatly reduce HTTP traffic to the data center network. HTTP provides a connectionless service. For each HTML element, a separate HTTP request must be sent to the mail or calendar service. These requests can be for static data, such as an image, style sheets, JavaScript files, or HTML files. By placing these elements closer to the end user, you reduce network traffic on the back-end data center.

Horizontal Scalability Strategy

Scalability is critical to organizations needing to make the most cost-effective use of their computing resources, handle peak workloads, and grow their infrastructure as rapidly as their business grows. Keep these points in mind:

- How the system responds to increasing workloads: what performance it provides, and as the workload increases, whether it crashes or enables performance to gracefully degrade.
- How easy it is to add processors, CPUs, storage, and I/O resources to a system or network to serve increasing demands from users.
- Whether the same environment can support applications as they grow from Low-end systems to mid-range servers and mainframe-class systems.

When deployed in a two-tiered architecture, the Communications Services offering is meant to scale very effectively in a horizontal manner. Each functional element can support increased load by adding additional machines to a given tier.

Scaling Front-end and Back-end Services

Note: This section will be rewritten for clarity.
In practice, the method for scaling the front-end and back-end services differs slightly. For Tier 1 elements, you start the scaling process when traffic to the front end grows beyond current capacity. You add relatively low cost machines to the tier and load balance across these machines. Thus, load balancers can precede each of the Tier 1 service functions as overall system load, service distribution, and scalability requirements dictate.

For Tier 2 elements, you start the scaling process when the back-end services have exceeded user or data capacity. As a general rule, design the Tier 2 services to accommodate just under double the load capacity of the Tier 1 services. For example, for an architecture designed for 5,000 users, the Tier 1 front-end services are designed to support 5,000 users. The back-end services are then doubled, and designed to accommodate 10,000 users. If the system capacity exceeds 5,000 users, the front-end services can be horizontally scaled. If the overall capacity reaches 5,000 users, then the back-end services can be scaled to accommodate. Such design enables flexibility for growth, whether the growth is in terms of users or throughput.

Implementing Local Message Transfer Protocol (LMTP) for Messaging Server

Best practices say you should implement LMTP to replace SMTP for message insertion. An LMTP architecture is more efficient for delivering to the back-end Message Store because it:

- Reduces the load on the stores. Relays are horizontally scalable while stores are not, thus it is a good practice to make the relays perform as much of the processing as possible.
- Reduces IOPS by as much as 30 percent by removing the MTA queues from the stores.
- Reduces the load on LDAP servers.
- The LDAP infrastructure is often the limiting factor in large messaging deployments.
- Reduces the number of message queues.
- Requires a small amount of shared/NFS storage across all inbound MTAs.

You need a two-tiered architecture to implement LMTP.

Background: "How Email Works" Introduction to Messaging Server

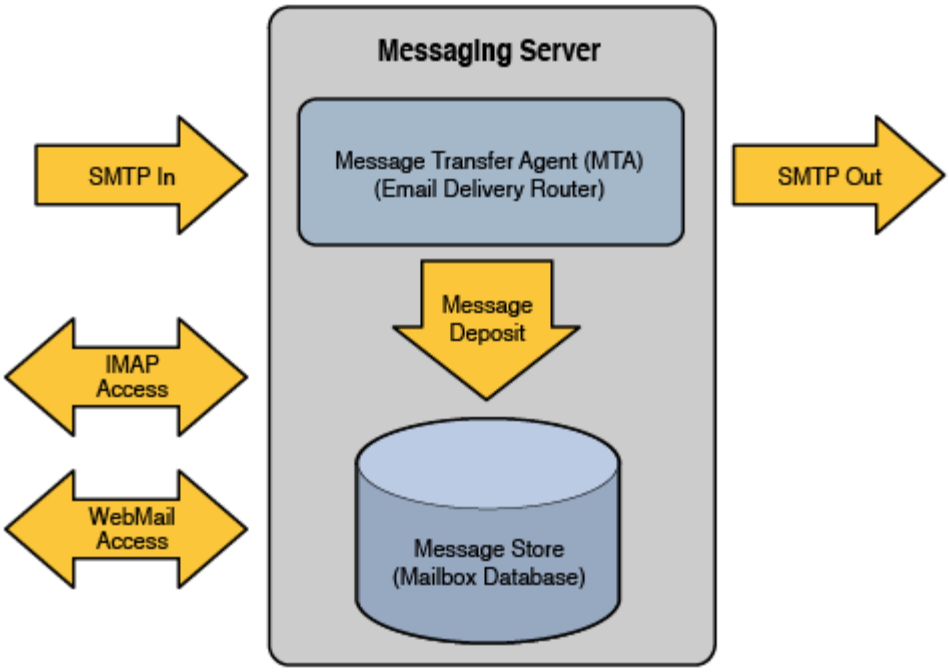
This write up is a basic introduction. It does avoid most of the more complicate configurations and mechanisms which are part of the Messaging Server. The key objective is to understand "how email works" from a basic "black box" model. In the up and coming posts, I'll start getting more complicated into the internals of the Messaging Server and the other components of the Communications Suite.

What Does Messaging Server Enable Users to Do?

The Messaging Server enables users to interact with their email message data in three different manners:

1. Users can write an email, and then send it to someone(s).
2. Users can receive emails from others.
3. Users can access their mailbox

Figure 3 How Messaging Server Allows Users to Interact with Message Data



The key components of Messaging Server are:

- Message Transport Agent (MTA)
- Message Store Database
- Message Access Services:
- IMAP / POP3
- WebMail Access for Communications Express (separate product)

The above figure simplifies the Messaging Server into the following three components.

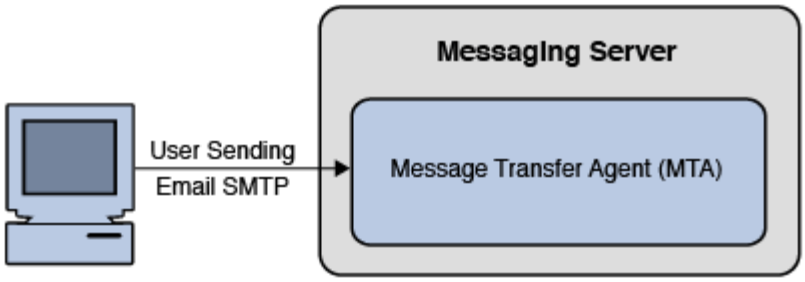
- Messages are delivered into the Messaging Server via the MTA. The MTA is the SMTP Server which accepts the email message and then route the message to it's destination. The destination for the email could be either the Message Store or to another server somewhere out over the network.
- The Message Store is an Oracle private database, which stores user's mailboxes and message data.
- Users access their mailboxes via either IMAP, POP3 or through Webmail.

A User Decides to Send an Email

Users write their emails using any of the popular email clients out on the market. These email clients could include Mozilla/Thunderbird, Apple Mail, Outlook Express, or any other IMAP-based email application. Once the email is ready, the user would click the "SEND" button.

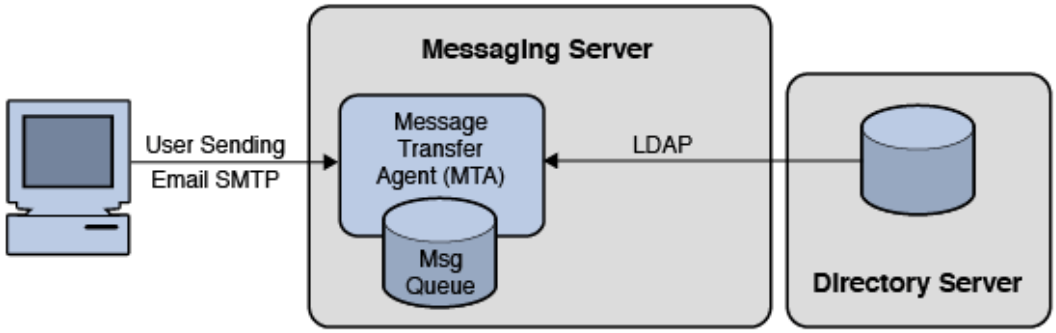
The "SEND" button allows the email application to connect with the MTA. The client connects to the MTA using the SMTP protocol.

Figure 4 Client Connects to the Messaging Server MTA



On the connection to the MTA, the client will send the message over the network to the MTA. The MTA would then store the message in the Message Queue Disk and release the client connection. The message stored on the queue will then be read and the address of the message will be evaluated by using information stored in the Directory Server using LDAP.

Figure 5 MTA Stores Message to Message Queue and Message Evaluated Through LDAP



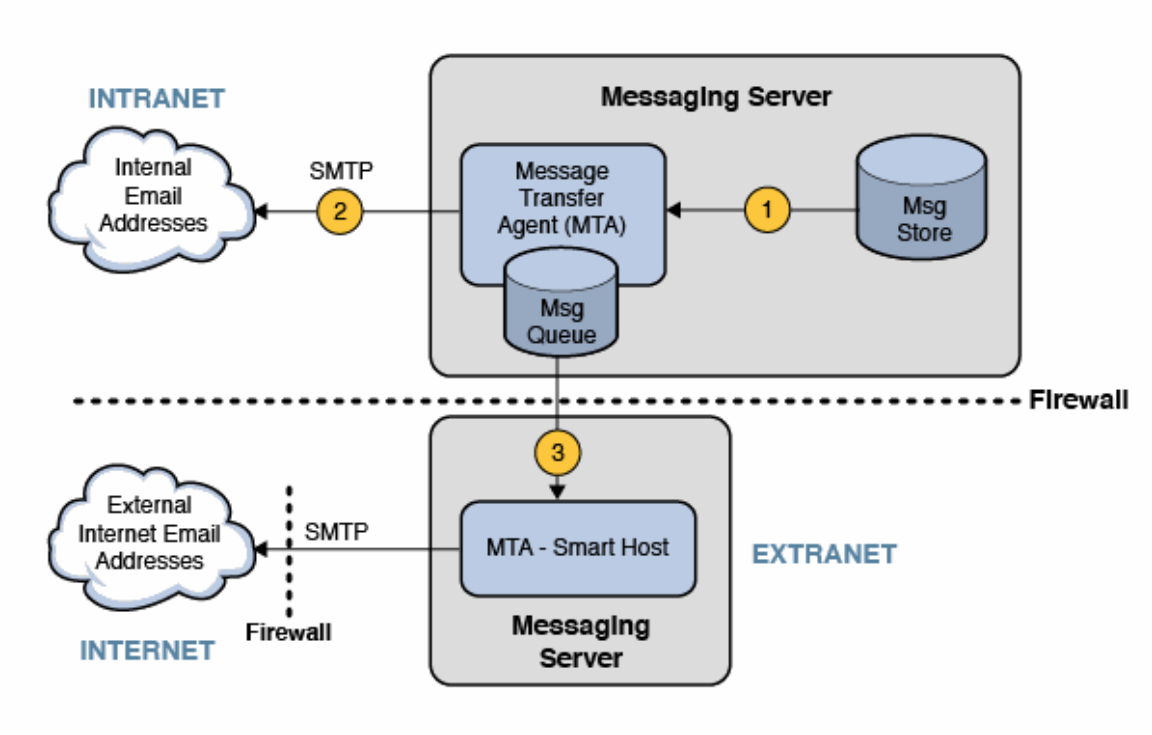
At this point, the MTA will attempt to deliver the message to one of three places:

1. Into a User's Mailbox
2. Into another Messaging Server
3. Into another site on the Internet (note for security.. it is recommend that the MTA forward the message to a "smarthost" MTA to handle the message delivery over the Internet)

It is important to note that once the message is on the Message Disk Queue, and the MTA has responsibility of the message. We guarantee this responsibility by committing the message to disk. It is this commitment which provides the first clue into a our first performance limitation. We write all MTA data to a disk queue, and we therefore are bounded by the disk i/o performance of that disk system. We also need to ensure that the disk system is highly available through RAID. (Typically RAID 0+1 or RAID 5.)

Postal Service Example: An example of this would be a letter that you drop into a postal mailbox. If you drop a letter of at the Post Office, you are assuming that the Postal Service will mail the message to the destination. You will not expect the message to get lost.
The MTA will not remove the message from the Message Disk Queue until a successful delivery. If a delivery attempt fails, it will keep trying. If it fails a final time, the message will be HELD on the disk queue and a notification sent back to the sender. The MTA logs will record the whole history.

Figure 6 How the MTA Delivers a Message



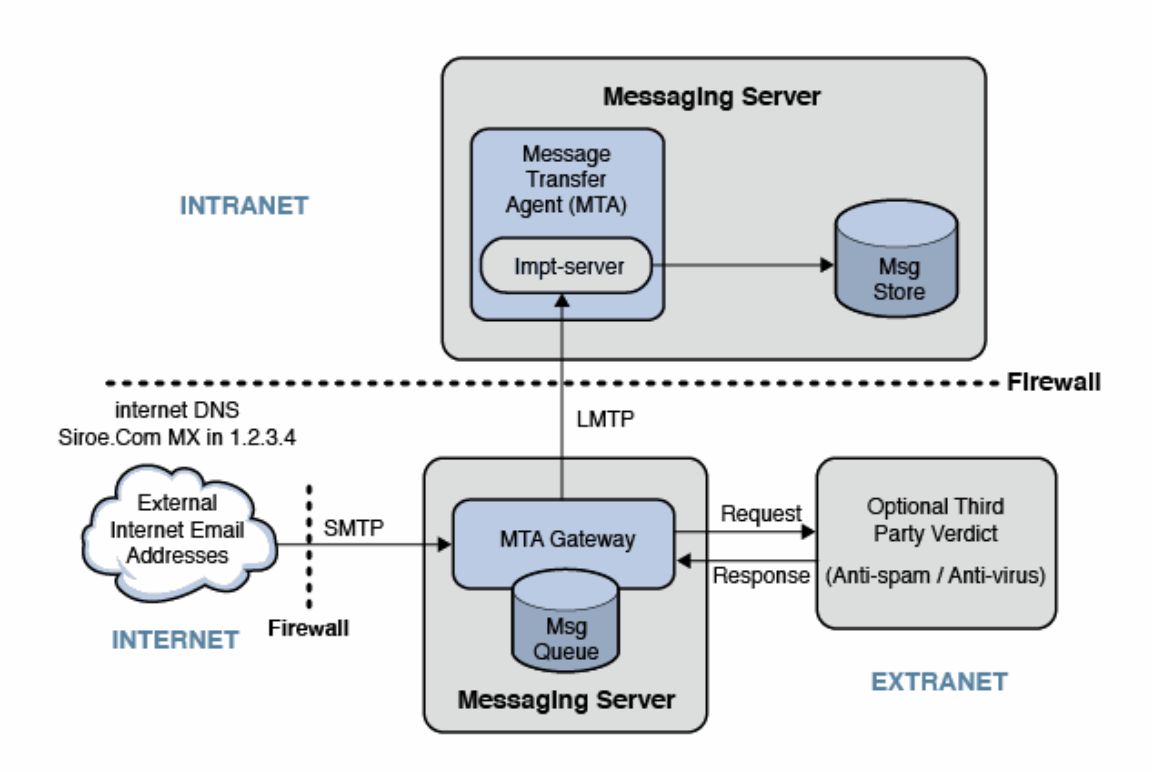
The MTA will attempt to deliver the message in one of three possible ways:

1. Message is Deposited into the User's Mailbox on the Message Store
2. Message is Sent to another MTA Server on the Same Local Network
3. Message is sent to the Internet by first sending to the smarthost.

The Smarthost MTA sits between the Messaging Server inside the protected network and the Internet. Typically this would be in an area outside the internal network. The Smarthost would evaluate the message address by domain name. Using DNS, the MTA would attempt to identify the DNS MX Record and A Record for the destination of the email. It would then attempt to deliver the message to the host of that IP Address using SMTP.

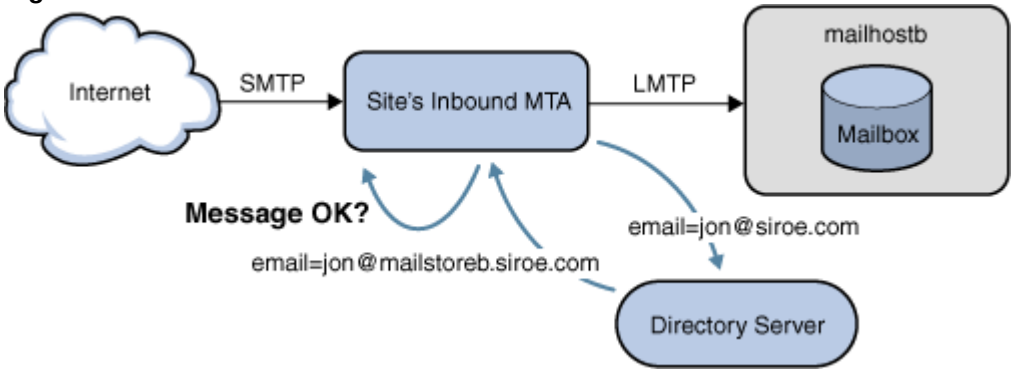
User Receives an Email

Figure 7 How a User Receives Email from the MTA



An email sent to a user on the Message Store. In the case of the message arriving from the Internet, the email sender would address the email to "jon@siroe.com". The email sender's MTA would look up "siroe.com" in DNS and find either an MX record or an A record for this site.

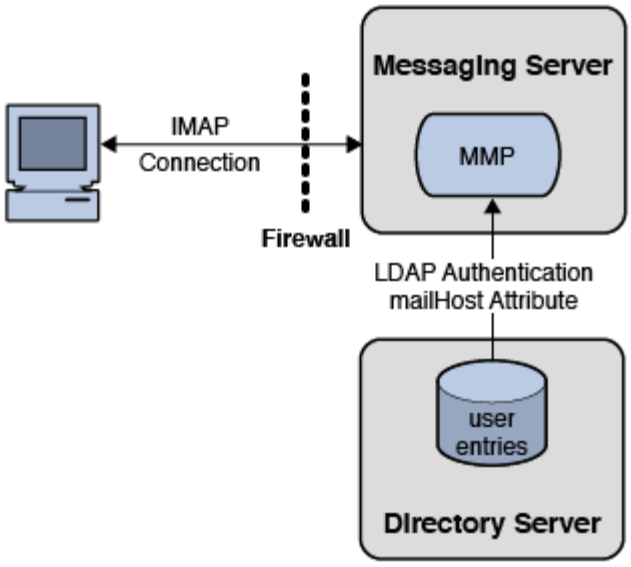
Figure 8 Inbound MTA



The Inbound MTA for the site may be configured to detect SPAM or Viruses. In this case, the message will be sent to a anti-spam/anti-virus verdict engine (such as Cloudmark or Symantec Brightmail). If the email passes, it is evaluated for message routing. The email address is evaluated in the Directory Server. If the email address is valid, it is then sent to the user's mailbox host (mailboxb).

User Access Mailbox

Figure 9 User Access to Mailbox

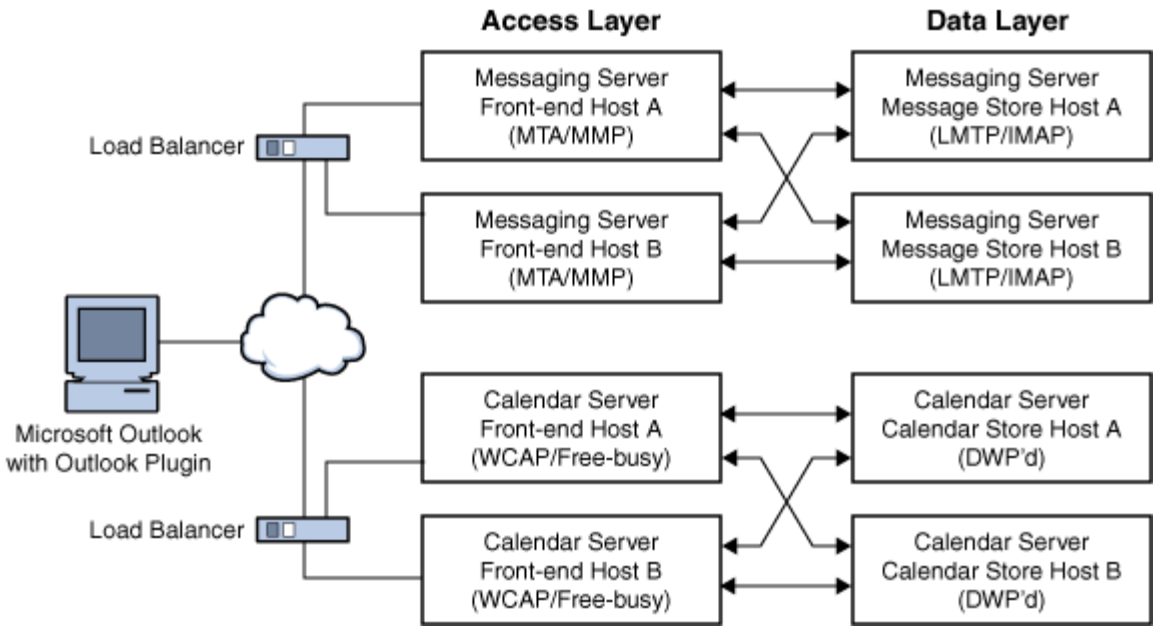


If users want to access their email, they use their email client. These clients, such as Mozilla Thunderbird or Apple Mail, connect to the mailbox using the IMAP protocol. In some cases, ISPs limit access to POP3 protocol.

The email client would connect to Messaging Server's Mail Multiplexor Proxy (MMP). The MMP would accept the user's email credentials (their username and password) and validate their login using the Directory Server. Should the authentication be successful, the MMP would then identify the user's backend Message Store server. The MMP would then connect to this backend Message Store on the user's behalf. The user can then access their mailbox data.

The value of the MMP is that it provides 1) Security and 2) Scalability. Security is improved by the use of the MMP by isolating the Message Store from the end-users. This helps to prevent unauthorized access (hacking) into the Message Store server data. Scalability (as seen in the picture below) is improved through Horizontal Scalability. This means that the architecture can grow by adding additional MMPs and backend Message Store servers.

Figure 10 Growing Access Layer Capacity


Access Layer
Data Layer
Load Balancer
Messaging Server
Front-end Host A
(MTA/MMP)
Messaging Server
Message Store Host A
(LMTP/IMAP)
Messaging Server
Front-end Host B
(MTA/MMP)
Messaging Server
Message Store Host B
(LMTP/IMAP)
Microsoft Outlook
with Outlook Plugin
Calendar Server
Front-end Host A
(WCAP/Free-busy)
Calendar Server
Calendar Store Host A
(DWP'd)
Load Balancer
Calendar Server
Front-end Host B
(WCAP/Free-busy)
Calendar Server
Calendar Store Host B
(DWP'd)

Configuring Message Expiration

Configuring Messaging Expiration (Tasks)

This information describes the tasks you use to expire messages. See Message Store Message Expiration for overview and conceptual information.

Topics:

- To Set imexpire Rules Textually
- To Set imexpire Folder Patterns
- To Schedule Message Expiration and Logging Level
- To Exclude Specified Users from Message Expiration

See also:

- Expiring Messages by Message Type

To Set imexpire Rules Textually

You expire messages by specifying rules in a `store.expirerule` file. The `store.expirerule` file contains one expire criteria per line. An expire criteria of the global rule configuration file *msg-svr-base* `/config/store.expirerule`) has the following format:

rule_name`.`*attribute*`:` *value*

An expiration rule for a user or mailbox rule configuration file has this format:

attribute`:` *value*

The following example shows a set of global expiration rules in the *msg-svr-base* `/config/store.expirerule` file.

Example imexpire Rules

```
Rule1.regexp: 1
Rule1.folderpattern: user/.*
Rule1.messagesize: 100000
Rule1.messagesizedays: 3
Rule1.deleted: or
Rule1.Subject: Vigara Now!
Rule1.Subject: XXX Porn!
Rule1.messagecount: 1000
Rule1.messagedays: 365
Rule2.regexp: 1
Rule2.folderpattern: user/.*@siroe.com/.*
Rule2.exclusive: 1
Rule2.deleted: or
Rule2.messagedays: 14
Rule2.messagecount: 1000
Rule3.folderpattern: user/f.dostoevski/inbox
Rule3.Subject: *On-line Casino*
```

Rule 1 sets the global expiration policy (that is, policy that applies to all messages) to:

- Enable UNIX regular expressions in rules creation.
- Remove messages larger than 100,000 bytes after 3 days.
- Remove messages deleted by the user.
- Remove any message with the strings "Vigara Now!" or "XXX Porn!" in the Subject: header.
- Limit all folders to 1,000 messages. After 1,000 messages, the system removes the oldest messages on a folder to keep the total to 1,000.
- Remove all messages older than 365 days.

Rule 2 sets the message expiration policy for users at the hosted domain `siroe.com`. It limits mailbox sizes to 1 megabyte, removes messages that have been deleted, and removes messages older than 14 days.

Rule 3 sets the message expiration policy for messages in the `inbox` folder of user `f.dostoevski`. It removes messages with a subject line having the expression "On-line Casino."

To Set imexpire Folder Patterns

Folder patterns can be specified by using POSIX regular expressions by setting the `imexpire` attribute `regex` to 1. If not specified, IMAP expressions are used. The format must start with a `user/` followed by a pattern. The following table shows the folder pattern for various folders.

imexpire Folder Patterns Using Regular Expressions

Scope	Folder Pattern (regex=0)	Folder Pattern (regex=1)
Apply rule to all messages in all folders of *userid*.	`user/`*`userid`*`/*`	`user/`*`userid`*`/.*`
Apply rule to messages of *userid* in folder `Sent`:.	`user/`*`userid`*`/Sent`	`user/`*`userid`*`/Sent`
Apply rule to entire message store.	`user/*`	`user/.*`
Apply rule to any folder called `Trash` anywhere in any user's hierarchy.	`user/*/Trash`	`user/.*/Trash`
Apply rule to folders in hosted domain `siroe.com`	`user/*@siroe.com/*`	`user/.*@siroe.com/.*`
Apply rule to folders in default domain.	Not applicable	`user/[^@]*/.*`

To Schedule Message Expiration and Logging Level

You activate message expiration by using the `imsched` scheduling daemon. By default, `imsched` invokes `imexpire` at 23:00 every day. In Messaging Server 6, `imsched` performed both expunge and purge. Starting with Messaging Server 7, the purge function has been moved to the impurge daemon. The `imexpire` schedule can be customized by setting the `configutil` parameter `local.schedule.expire` as described in Expire and Purge configutil Log and Scheduling Parameters .

Expire and purge can take a long time to complete on a large message store. You should experiment and decide how often to run these processes. For example, if an expire and purge cycle takes 10 hours, you might not want the default schedule of running expire and purge once a day. Schedule expire and purge by using the `imexpire` command and the automatic task scheduling parameter (see To Schedule Automatic Tasks). For example:

```
configutil -o local.schedule.expire -v "0 1 * * 6 bin/imexpire -e"
```

The following command applies to **Messaging Server 6.3**. The command was removed in **Messaging Server 7**. Beginning with **Messaging Server 7**, the `impurge` command is enabled and starts when Messaging Server is started.

```
configutil -o local.schedule.mspurge -v "0 23 * * * bin/imexpire -c"
```

In this example, messages are expired at 1 AM Saturdays and purged every night at 11 PM. If no purge schedule is set, `imexpire` performs purge after an expire.

Expire and Purge configutil Log and Scheduling Parameters

For Purge options, see Maintenance Queue configutil Parameters.

Expire and Purge configutil Log and Scheduling Parameters

Parameter	Description
`local.schedule.expire.enable`	Whether the expire task should be scheduled. Default: 1

`local.schedule.expire`	Interval for running `imexpire`. Uses UNIX `crontab` format:
	minute hour day-of-month month-of-year day-of-week
	
	The values are separated by a space or tab and can be 0-59, 0-23, 1-31, 1-12 and 0-6 (with 0=Sunday) respectively. Each time field can be either an asterisk (meaning all legal values), a list of comma-separated values, or a range of two values separated by a hyphen. Note that days can be specified by both day of the month and day of the week, however, it is not typical to use them both since the number of such occurrences are very small. If they are both specified, then both will be required. For example, setting the 17th day of the month and Tuesday will require both values to be true.
	
	You can also use the `-e` and `-c` flags with `imexpire` to and expire only or purge only respectively. See `imexpire`.
	
	Interval Examples:
	
	1) Run `imexpire` at 12:30am, 8:30am, and 4:30pm:
	`30 0,8,16 * * * bin/imexpire`
	
	2) Run `imexpire` at weekday morning at 3:15 am:
	`15 3 * * 1-5 bin/imexpire`
	
	3) Run `imexpire` only on Mondays:
	`0 0 * * 1 bin/imexpire`
	
	Default:
	`0 23 * * * bin/imexpire`
	
	To disable: set `local.schedule.expire.enable` to `NO`.

`local.store.expire.loglevel`	Specify a log level: 1 = log summary for the entire expire session. 2 = log one message per mailbox expired. 3 = log one message per message expired. Default: 1

To Set imexpire Logging Levels

`imexpire` logs a summary to the default log file upon completion. If expire is invoked from the command line, the `-v` (verbose) and `-d` (debug) options can be used to instruct `imexpire` to log detail status and debug messages to `stderr`. If `imexpire` is invoked by `imsched`, the `configutil` parameter `local.store.expire.loglevel` can be set to 1, 2, or 3 for different levels of logging. Loglevel 1 is the default, and logs a summary for the entire expire session. Loglevel 2 logs one message per mailbox expired. Loglevel 3 logs one message per message expired.

The following example invokes expire from the command line to set verbose logging and shows the resulting messages in the default log file, *msg-svr-base*`/log/default`.

```
# cd /opt/sun/comms/messaging64/sbin
# ./imexpire -n -v 1
# tail ../log/default
[25/Nov/2010:11:51:51 +1100] server imexpire[20730]: General Notice:
imexpire started
[25/Nov/2010:11:51:51 +1100] server imexpire[20730]: General Notice:
iBiff plugin loaded: ms-internal
[25/Nov/2010:11:51:51 +1100] server imexpire[20730]: General Notice:
primary partition: expired 0 messages
[25/Nov/2010:11:51:51 +1100] server imexpire[20730]: General Notice:
Expired 0 messages
[25/Nov/2010:11:51:51 +1100] server imexpire[20730]: General Notice:
Expire finished
```

To Exclude Specified Users from Message Expiration

Exclude specified users from the expire rules by adding their user ID, one per line, in a file called `expire_exclude_list` in the *msg-svr-base*`/config` directory. Or, configure a "dummy" exclusive expire rule under the user's mailbox. For example,

```
# cat
/opt/sun/comms/messaging64/data/store/partition/primary/=user/<nn>/<nn>/=<u
1
```

Configuring POP, IMAP, and HTTP Services

Configuring POP, IMAP, and HTTP Services

Messaging Server supports the Post Office Protocol 3 (POP3), the Internet Mail Access Protocol 4 (IMAP4), and the HyperText Transfer Protocol (HTTP) for client access to mailboxes. IMAP and POP are both Internet-standard mailbox protocols. Oracle Communications Convergence, a web-enabled electronic mail program, enables end users to access their mailboxes by using a browser running on an Internet-connected computer system using HTTP.

> **Note**
> Starting with Messaging Server 7, the Messenger Express client interface has been removed. The HTTP (Webmail) service is still used by the Communications Express and Convergence interfaces to access user email from the message store.

This information describes how to configure your server to support one or more of these services by using command-line utilities.

Topics:

General Configuration

Configuring the general features of the Messaging Server POP, IMAP, and HTTP services includes enabling or disabling the services, assigning port numbers, and optionally modifying service banners sent to connecting clients. This section provides background information. For the steps you follow to make these settings, see To Configure POP Services, To Configure IMAP Services and To Configure HTTP Services.

This section consists of the following subsections:

Enabling and Disabling Services

You can control whether any particular instance of Messaging Server makes its POP, IMAP, or HTTP service available for use. This is not the same as starting and stopping services (see Starting and Stopping Services). To function, POP, IMAP, or HTTP must be both enabled and started.

Enabling a service is a more "global" process than starting or stopping a service. For example, the Enable setting persists across system reboots, whereas you must restart a previously "stopped" service after a reboot.

There is no need to enable services that you do not plan to use. For example, if a Messaging Server instance is used only as a Mail Transfer Agent (MTA), you should disable POP, IMAP, and HTTP services. If a Messaging Server instance is used only for POP services, you should disable IMAP and HTTP. If a Messaging Server instance is used only for web-based email, you should disable both POP and IMAP.

You can enable or disable services at the server level, described later in this information. To Specify What Services Can Be Started also describes this process. In addition, you can enable or disable services at the user level by setting the LDAP attribute `mailAllowedServiceAccess`.

Specifying Port Numbers

For each service, you can specify the port number that the server is to use for service connections:

- If you enable the POP service, you can specify the port number that the server is to use for POP connections. The default is 110.
- If you enable the IMAP service, you can specify the port number that the server is to use for IMAP connections. The default is 143.
- If you enable the HTTP service, you can specify the port number that the server is to use for HTTP connections. The default is 8990.

You might need to specify a port number other than the default if you have, for example, two or more IMAP server instances on a single host machine, or if you are using the same host machine as both an IMAP server and a Messaging Multiplexor server. (For information about the Multiplexor, see Configuring and Administering Multiplexor Services.

Keep the following in mind when you specify a port:

- Port numbers can be any number from 1 to 65535.
- Make sure the port you choose isn't already in use or reserved for another service.

Ports for Encrypted Communications

Messaging Server supports encrypted communications with IMAP, POP, and HTTP clients by using the Secure Sockets Layer (SSL) protocol. For general information on support for SSL in Messaging Server, see Configuring Encryption and Certificate-Based Authentication.

IMAP Over SSL

You can accept the default (recommended) IMAP over SSL port number (993) or you can specify a different port for IMAP over SSL.

Messaging Server provides the option of using separate ports for IMAP and IMAP over SSL because most current IMAP clients require separate ports for them. Same-port communication with both IMAP and IMAP over SSL is an emerging standard. As long as your Messaging Server has an installed SSL certificate (see Obtaining Certificates), it can support same-port IMAP over SSL.

POP Over SSL

The default separate SSL port for POP is 995. You can also initiate SSL over normal POP port with the command "STLS" (see To Configure POP Services).

HTTP Over SSL

You can accept the default HTTP over SSL port number (8991) or you can specify a different port for HTTPS.

Service Banner

When a client first connects to the Messaging Server POP or IMAP port, the server sends an identifying text string to the client. This service banner (not normally displayed to the client's user) identifies the server as Messaging Server, and gives the server's version number. The banner is most typically used for client debugging or problem-isolation purposes.

You can replace the default banner for the POP or IMAP service if you want a different message sent to connecting clients.

Use the `configutil` utility and the (`service.pop.banner`) parameter to set service banners. For detailed syntax information about `configutil`, see http://msg.wikidoc.info/index.php/Configutil_Reference.

Login Requirements

You can control how users are permitted to log in to the POP, IMAP, or HTTP service to retrieve mail. You can allow password-based login (for all services), and certificate-based login (for IMAP or HTTP services). This section provides background information. For the steps you follow to make these settings, see To Configure POP Services, To Configure IMAP Services or To Configure HTTP Services. In addition, you can specify the valid login separator for POP logins. This section consists of the following subsections:

- To Set the Separator for POP Clients
- To Allow Log In without Using the Domain Name
- Password-Based Login
- Certificate-Based Login

To Set the Separator for POP Clients

Some mail clients will not accept @ as the login separator (that is, the @ in an address like `uid@domain`). Examples of these clients are Netscape Messenger 4.76, Netscape Messenger 6.0, and Microsoft Outlook Express on Windows 2000. The workaround is as follows:

1. Make + a valid separator with the following command:
   `configutil -o service.loginseparator -v "@+"`
2. Inform POP client users that they should login with + as the login separator, not @.

To Allow Log In without Using the Domain Name

A typical login involves the user entering a user ID followed by a separator and the domain name and then the password. Users in the default domain specified during installation, however, can log in without entering a domain name or separator.

To allow users of other domains to log in with just the user ID (that is, without having to use the domain name and separator) set `sasl.default.ldap.searchfordomain` to **`0`**. The user ID must be unique to the entire directory tree. If it is not unique, logging in without the domain name will not work.

You might want to modify the attribute that user must enter to log in. For example, to allow the user to log in with a phone number (`telephoneNumber`) or employee number (`employeeID`), change the LDAP search defined by the `configutil` parameter `sasl.default.ldap.searchfilter`. This parameter is a global default setting for the `inetDomainSearchFilter` per-domain attribute and follows the same syntax.

Refer to the http://msg.wikidoc.info/index.php/Configutil_Reference for further information on these parameters.

Password-Based Login

In typical messaging installations, users access their mailboxes by entering a password into their POP,

IMAP, or HTTP mail client. The client sends the password to the server, which uses it to authenticate the user. If the user is authenticated, the server decides, based on access-control rules, whether or not to grant the user access to certain mailboxes stored on that server.

If you allow password login, users can access POP, IMAP, or HTTP by entering a password. (Password- or SSL-based login is the only authentication method for POP services.) Passwords are stored in an LDAP directory. Directory policies determine what password policies, such as minimum length, are in effect.

If you disallow password login for IMAP or HTTP services, password-based authentication is not permitted. Users are then required to use certificate-based login, as described in the next section.

To increase the security of password transmission for IMAP and HTTP services, you can require that passwords be encrypted before they are sent to your server. You do this by selecting a minimum cipher-length requirement for login.

- If you choose 0, you do not require encryption. Passwords are sent in the clear or they are encrypted, depending on client policy.
- If you choose a nonzero value, the client must establish an SSL session with the server by using a cipher whose key length is at least the value you specify, thus encrypting any IMAP or HTTP user passwords the client sends.

If the client is configured to require encryption with key lengths greater than the maximum your server supports, or if your server is configured to require encryption with key lengths greater than what the client supports, password-based login cannot occur. For information on setting up your server to support various ciphers and key lengths, see To Enable SSL and Selecting Ciphers.

Certificate-Based Login

In addition to password-based authentication, Oracle servers support the authentication of users through examination of their digital certificates. Instead of presenting a password, the client presents the user's certificate when it establishes an SSL session with the server. If the certificate is validated, the user is considered authenticated.

For instructions on setting up Messaging Server to accept certificate-based user login to the IMAP or HTTP service, see To Set Up Certificate-Based Login.

If you have performed the tasks required to set up certificate-based login, both password-based and certificate-based login are supported. Then, if the client establishes an SSL session and supplies a certificate, certificate-based login is used. If the client does not use SSL or does not present a client certificate, it sends a password instead.

Performance Parameters

You can set some of the basic performance parameters for the POP, IMAP, and HTTP services of Messaging Server. Based on your hardware capacity and your user base, you can adjust these parameters for maximum efficiency of service. This section provides background information. For the steps you follow to make these settings, see To Configure POP Services, To Configure IMAP Services or To Configure HTTP Services.

This section consists of the following subsections:

- Number of Processes
- Number of Connections per Process
- Number of Threads per Process
- Dropping Idle Connections
- Logging Out HTTP Clients

Number of Processes

Messaging Server can divide its work among several executing processes, which in some cases can increase efficiency. This capability is especially useful with multiprocessor server machines, in which adjusting the number of server processes can allow more efficient distribution of multiple tasks among the hardware processors.

There is a performance overhead, however, in allocating tasks among multiple processes and in switching from one process to another. The advantage of having multiple processes diminishes with each new one added. A simple rule of thumb for most configurations is to have one `imapd` and one `popd` process per hardware processor on your server machine, up to a maximum of perhaps four processes. Your optimum configuration might be different. This rule of thumb is meant only as a starting point for your own analysis.

> **Note**
> On some platforms you might also want to increase the number of processes to get around certain per-process limits (such as the maximum number of file descriptors), specific to that platform, that might affect performance. The default number of processes is one each for the POP, IMAP, or HTTP service.

> **Note**
> For `mshttpd`, starting in Messaging Server 6.3, use the Messaging Server 64-bit version and only one `mshttpd` process. Increase the number of connections per process as needed.

Number of Connections per Process

The more simultaneous client connections your POP, IMAP, or HTTP service can maintain, the better it is for clients. If clients are denied service because no connections are available, they must then wait until another client disconnects.

On the other hand, each open connection consumes memory resources and makes demands on the I/O subsystem of your server machine, so there is a practical limit to the number of simultaneous sessions you can expect the server to support. (You might be able to increase that limit by increasing server memory or I/O capacity.)

IMAP, HTTP, and POP have different needs in this regard:

- IMAP connections are generally long-lived compared to POP and HTTP connections. When a user connects to IMAP to download messages, the connection is usually maintained until the user quits or the connection times out. In contrast, a POP or HTTP connection is usually closed as soon as the POP or HTTP request has been serviced.
- IMAP and HTTP connections are generally very efficient compared to POP connections. Each POP reconnection requires reauthentication of the user. In contrast, an IMAP connection requires only a single authentication because the connection remains open for the duration of the IMAP session (login to logout). An HTTP connection is short, but the user need not reauthenticate for each connection because multiple connections are allowed for each HTTP session (login to logout). POP connections, therefore, involve much greater performance overhead than IMAP or HTTP connections. Messaging Server, in particular, has been designed to require very low overhead by open but idle IMAP connections and by multiple HTTP connections.

> **Note**
> For more information about HTTP session security, see About HTTP Security.

Thus, at a given moment for a given user demand, Messaging Server may be able to support many more

open IMAP or HTTP connections than POP connections.

The default value for IMAP is 4000. The default value for HTTP is 6000 connections per process. The default value for POP is 600. These values represent roughly equivalent demands that can be handled by a typically configured server machine. Your optimum configuration might be different. These defaults are meant only as general guidelines.

Typically, active POP connections are much more demanding on server resources and bandwidth than active IMAP connections since IMAP connections are idle most of the time while POP connections are constantly downloading messages. Having a lower number of sessions for POP is correct. Conversely, POP connections only last as long as it takes to download email, so an active POP user is only connected a small percentage of the time, while IMAP connections stay connected between successive mail checks.

Number of Threads per Process

Besides supporting multiple processes, Messaging Server further improves performance by subdividing its work among multiple threads. The server's use of threads greatly increases execution efficiency, because commands in progress are not holding up the execution of other commands. Threads are created and destroyed, as needed during execution, up to the maximum number you have set.

Having more simultaneously executing threads means that more client requests can be handled without delay, so that a greater number of clients can be serviced quickly. However, there is a performance overhead to dispatching among threads, so there is a practical limit to the number of threads the server can make use of.

For POP, IMAP, and HTTP, the default maximum value is 250 threads per process. The numbers are equal despite the fact that the default number of connections for IMAP and HTTP is greater than for POP. It is assumed that the more numerous IMAP and HTTP connections can be handled efficiently with the same maximum number of threads as the fewer, but busier, POP connections. Your optimum configuration might be different, but these defaults are high enough that it is unlikely you would ever need to increase them; the defaults should provide reasonable performance for most installations.

Dropping Idle Connections

To reclaim system resources used by connections from unresponsive clients, the IMAP4, POP3, and HTTP protocols permit the server to unilaterally drop connections that have been idle for a certain amount of time.

The respective protocol specifications require the server to keep an idle connection open for a minimum amount of time. The default times are 10 minutes for POP, 30 minutes for IMAP, 3 minutes for HTTP. You can increase the idle times beyond the default values, but you cannot make them less.

If a POP or IMAP connection is dropped, the user must reauthenticate to establish a new connection. In contrast, if an HTTP connection is dropped, the user need not reauthenticate because the HTTP session remains open. For more information about HTTP session security, see About HTTP Security.

Idle POP connections are usually caused by some problem (such as a crash or hang) that makes the client unresponsive. Idle IMAP connections, on the other hand, are a normal occurrence. To keep IMAP users from being disconnected unilaterally, IMAP clients typically send a command to the IMAP server at some regular interval that is less than 30 minutes.

Logging Out HTTP Clients

An HTTP session can persist across multiple connections. HTTP clients are not logged out when a connection is dropped. However, if an HTTP session remains idle for a specified time period, the server will automatically drop the HTTP session and the client is logged out (the default time period is 2 hours).

When the session is dropped, the client's session ID becomes invalid and the client must reauthenticate to establish another session. For more information about HTTP security and session ID's, see About HTTP Security.

Client Access Controls

Messaging Server includes access-control features that enable you to determine which clients can gain access to its POP, IMAP, or HTTP messaging services (and SMTP as well). You can create flexible access filters that allow or deny access to clients based on a variety of criteria.

Client access control is an important security feature of Messaging Server. For information on creating client access-control filters and examples of their use, see Configuring Client Access to POP, IMAP, and HTTP Services. For information about configuring client access to SMTP services, see Mail Filtering and Access Control.

To Configure POP Services

You configure the Messaging Server POP service by using the `configutil` command. This section lists the more common POP services options. http://msg.wikidoc.info/index.php/Configutil_Reference provides a complete listing of options.

> **Note**
> For the POP service, password-based login is automatically enabled.

For more information, see also:

- Enabling and Disabling Services
- To Set the Login Separator for POP Clients
- Specifying Port Numbers
- Number of Connections per Process
- Dropping Idle Connections
- Number of Threads per Process
- Number of Processes

- To enable or disable the POP service:
  `configutil -o service.pop.enable -v [ yes | no ]`

- To specify the port number:
  `configutil -o service.pop.port -v` *number*

- To set the maximum number of network connections per process (see Number of Connections per Process for details):
  `configutil -o service.pop.maxsessions -v` *number*

- To set the maximum idle time for connections (see Dropping Idle Connections for details):
  `configutil -o service.pop.idletimeout -v` *number*

- To set the maximum number of threads per process (see Number of Threads per Process for more information):
  `configutil -o service.pop.maxthreads -v` *number*

- To set the maximum number of processes (see Number of Processes for additional information):
  `configutil -o service.pop.numprocesses -v` *number*

- To enable POP over SSL on port 995:

```
# ./configutil -o service.pop.enablesslport -v 1
# ./configutil -o service.pop.sslusessl -v 1
# ./configutil -o service.pop.sslport -v 995
# ./stop-msg pop
# ./start-msg pop
```

TLS is also supported if SSL is configured correctly.

- To specify a protocol welcome banner:
  `configutil -o service.pop.banner -v` *banner*

To Configure IMAP Services

You configure the Messaging Server IMAP service by using the `configutil` command. This section lists the common IMAP services options. http://msg.wikidoc.info/index.php/Configutil_Reference provides a complete listing of options. For more information, see also:

- Enabling and Disabling Services
- Specifying Port Numbers
- Password-Based Login
- Number of Connections per Process
- Dropping Idle Connections
- Number of Threads per Process
- Number of Processes
- Configuring IMAP IDLE

Command Line: You can set values for the IMAP attributes at the command line as follows:

- To enable or disable the IMAP service:
  `configutil -o service.imap.enable -v [ yes | no ]`

- To specify the port number:
  `configutil -o service.imap.port -v` *number*

- To enable a separate port for IMAP over SSL:
  `configutil -o service.imap.enablesslport -v [ yes | no ]`

- To specify a port number for IMAP over SSL:
  `configutil -o service.imap.sslport -v` *number*

- To enable or disable password login to the IMAP service:
  `configutil -o service.imap.plaintextmincipher -v` *value*
  If *value* is greater than 0, disable use of plaintext passwords unless a security layer (SSL or TLS) is activated. This forces users to enable SSL or TLS on their client to log in, which prevents exposure of their passwords on the network. Default is 0.

- To set the maximum number of network connections per process (see Number of Connections per Process for additional information):
  `configutil -o service.imap.maxsessions -v` *number*

- To set the maximum idle time for connections (see Dropping Idle Connections for additional information):
  `configutil -o service.imap.idletimeout -v` *number*

- To set the maximum number of threads per process (see Number of Threads per Process):
  `configutil -o service.imap.maxthreads -v` *number*

- To set the maximum number of processes (see Number of Processes):
  `configutil -o service.imap.numprocesses -v` *number*
- To specify a protocol welcome banner:
  `configutil -o service.imap.banner -v` *banner*
- To enable IMAP over SSL on port 993:

```
# ./configutil -o service.imap.enablesslport -v 1
# ./configutil -o service.imap.sslusessl -v 1
# ./configutil -o service.imap.sslport -v 993
# ./stop-msg imap
# ./start-msg imap
```

Configuring IMAP IDLE

The IMAP IDLE extension to the IMAP specification, defined in RFC 2177, enables an IMAP server to notify the mail client when new messages arrive and other updates take place in a user's mailbox. See Configuring IMAP IDLE for conceptual and task information on enabling IMAP IDLE in Messaging Server.

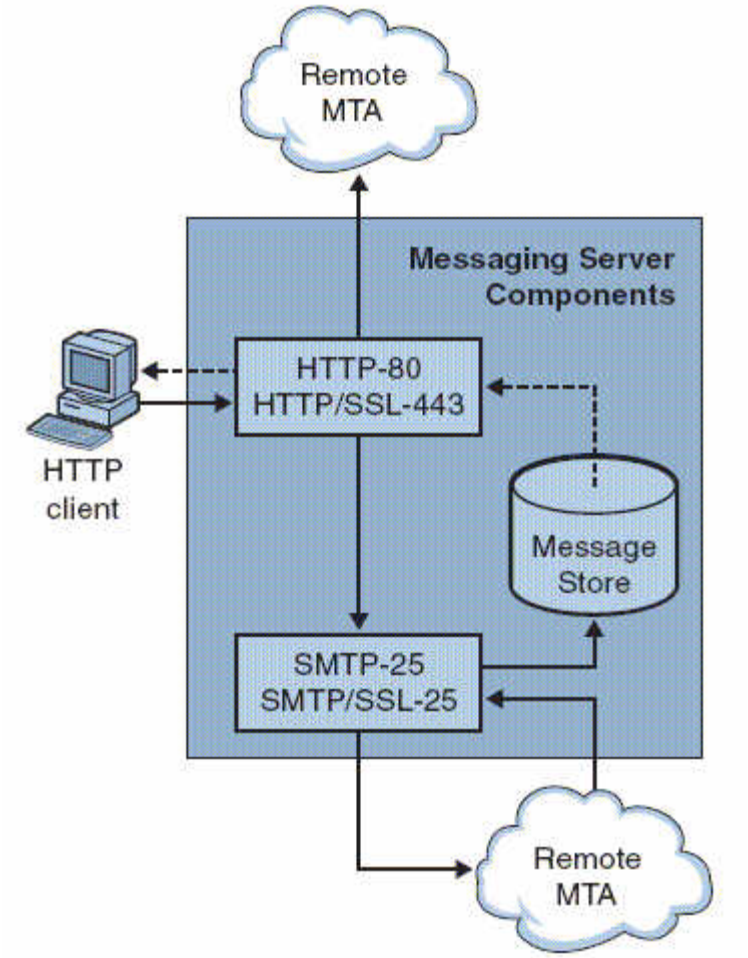
To Configure HTTP Services

Starting with **Messaging Server 7**, Messaging Server supports the HTTP mail clients Convergence and Communications Express.

Note
Starting with Messaging Server 7, the Messenger Express client interface has been removed.

While POP and IMAP clients send mail directly to a Messaging Server MTA for routing or delivery, HTTP clients send mail to a specialized web server called the Webmail Server (also called `mshttpd` or Messaging Server HTTP daemon). Depending on where the message is addressed, the Webmail Server directs the mail to an outbound MTA for routing or to one of the back-end message stores using IMAP, as shown in the following figure. The Communications Express Server simply routes requests to and from the Webmail Server.

HTTP Service Components



In previous versions, the Webmail Server accessed the message store directly. Now, it accesses the message store through the IMAP server. This provides several advantages:

- Messenger Express and Communications Express clients are now able to access shared folders that are located on different back-end message stores.
- The Webmail Server no longer must be installed on each back-end server.
- The Webmail Server can serve as a front-end server performing the multiplexing capabilities previously performed by Messenger Express Multiplexor (MEM).
- The Messenger Express Multiplexor (MEM) is obsolete and is no longer used.
- On the client side, nothing is changed except that users can now access shared folders that are not on their message store.

In previous versions, the MEM received and forwarded HTTP client requests to the appropriate Webmail Server on the back-end message store. Because of this, a copy of `mshttpd` had to be installed on every back-end server. Now, the Webmail Server operates as a front-end server receiving HTTP client email requests. It translates these requests to SMTP or IMAP calls and forwards the calls to either the MTA or the appropriate IMAP server on the back-end message store. If Messaging Server is used only for web-based email, make sure that IMAP is enabled.

Configuring Your HTTP Service

Many of the HTTP configuration parameters are similar to the parameters available for the POP and IMAP services, including parameters for connection settings and process settings. This section lists common HTTP service options. http://msg.wikidoc.info/index.php/Configutil_Reference provides a complete listing of options. For more information, see also:

- Enabling and Disabling Services

- Specifying Port Numbers
- Password-Based Login
- Number of Connections per Process
- Dropping Idle Connections
- Logging Out HTTP Clients
- Number of Threads per Process
- Number of Processes

For each IMAP server that users access, the Webmail Server needs to know the IMAP port, whether to use SSL, and the administrative credentials to use for user log-in. The `configutil` parameters to do this are as follows:

`local.service.proxy.imapport[.`*hostname*`]` – IMAP port on which to connect (default 143).

`local.service.proxy.imapssl` – Enable SSL (default no).

`local.service.proxy.admin[.`*hostname*`]` – Admin ID.

`local.service.proxy.adminpass[.`*hostname*`]` – Admin password.

You can set these parameters globally (applying to every IMAP back-end server), or for each individual IMAP back-end server, by appending the back end's fully qualified domain name to the option name.

To use IMAP over SSL, you must configure `mshttpd` as an SSL HTTP server, and the `mshttpd` certificate database must trust the IMAP back end's CA. You must enable `service.http.sslusessl`. If the back-end message store running IMAP is using a self-signed certificate (for example, as created by `generate-certDB`), then this certificate needs to be added to the front-end `mshttpd` daemon server.

If `local.service.proxy.admin/pass` is not configured, logins are rejected. The system provides the error message, "`Mail server unavailable. Administrator, check server log for details`" and the HTTP log lists the missing configuration options.

Additional values for HTTP attributes can be set at the command line as follows:

- To enable or disable the HTTP service:
  `configutil -o service.http.enable -v [ yes | no ]`

By default, the HTTP service sends outgoing web mail to the local MTA for routing or delivery. You might want to configure the HTTP service to send mail to a remote MTA, for example, if your site is a hosting service and most recipients are not in the same domain as the local host machine. To send web mail to a remote MTA, you need to specify the remote host name and the SMTP port number for the remote host.

- To specify the port number:
  `configutil -o service.http.port -v` *number*

- To enable a separate port for HTTP over SSL:
  `configutil -o service.http.enablesslport -v [ yes | no ]`

- To specify a port number for HTTP over SSL:
  `configutil -o service.http.sslport -v` *number*

- To enable or disable password login:
  `configutil -o service.http.plaintextmincipher -v` *value*
  If *value* is greater than 0, then disable use of plaintext passwords unless a security layer (SSL or TLS) is activated. This forces users to enable SSL or TLS on their client to log in, which prevents exposure of their passwords on the network. Default is 0.

- To set the maximum number of network connections per process (for more information, see Number of Connections per Process):

`configutil -o service.http.maxsessions -v` *number* (for more information, see Dropping Idle Connections)

- To set the maximum idle time for client sessions (for more information, see Logging Out HTTP Clients):
  `configutil -o service.http.sessiontimeout -v` *number*

- To set the maximum number of threads per process:
  `configutil -o service.http.maxthreads -v` *number*

- To set the maximum number of processes:
  `configutil -o service.http.numprocesses -v` *number*

When an HTTP client constructs a message with attachments, the attachments are uploaded to the server and stored in a file. The HTTP service retrieves the attachments and constructs the message before sending the message to an MTA for routing or delivery. You can accept the default attachment spool directory or specify an alternate directory. You can also specify a maximum size allowed for attachments. To specify the attachment spool directory for client outgoing mail use the following command. This includes all the attachments encoded in base64, and that base64 encoding requires an extra 33 percednt more space. Thus, a 5 Mbyte limit in the parameter results in the maximum size of one message and attachments being about 3.75 Mbyte.

- To set the spool directory:
  `configutil -o service.http.spooldir -v` *dirpath*

- To specify the maximum message size:
  `configutil -o service.http.maxmessagesize -v` *size*
  where *size* is a number in bytes. Note that this includes all the attachments encoded in base64, and that base64 encoding requires an extra 33 percent more space. Thus, a 5 Mbyte limit in the parameter results in the maximum size of one message and attachments being about 3.75 Mbytes.

- To specify an alternate MTA host name:
  `configutil -o service.http.smtphost -v` *hostname*

- To specify the port number for the alternate MTA host name:
  `configutil -o service.http.smtpport -v` *portnum*

To enable HTTP access over SSL on port 8991:

```
# ./configutil -o service.http.enablesslport -v 1
# ./configutil -o service.http.sslusessl -v 1
# ./configutil -o service.http.sslport -v 8991
# ./stop-msg http
# ./start-msg http
```

Managing Mailboxes

To Manage Mailboxes

This information describes how to list, create, remove, rename, move, and view information about mailboxes. It also describes how to find the directory location for a particular mailbox, restore expunged messages, and see how many users other than the mailbox owner have read messages in a shared IMAP folder. Specifically it describes the following utilities: `mboxutil`, `hashdir`, `readership` and consists of the following sections:

To Manage Mailboxes with mboxutil

Use the `mboxutil` command to perform typical maintenance tasks on mailboxes. `mboxutil` tasks include the following:

- List mailboxes
- List and remove orphaned and inactive mailboxes
- Create mailboxes
- Rename mailboxes
- Move mailboxes from one partition to another
- Expunge mailboxes
- Restore expunged messages that have not been purged
- List personal mailbox subscriptions and unsubscribed mailboxes that no longer exist
- You can also use the `mboxutil` command to view information about quotas. For more information, see Managing Message Store Quotas.

Starting with **Messaging Server 7 Update 4 Patch 26**, when an end user deletes a mailbox, all messages are expunged and purged according to the value of `store.cleanupage`. However, expunged messages can be restored by using the `mboxutil -R` command, as long as you have enabled the `store.mailboxpurgedelay` option. Expunged messages are moved to the new location when a mailbox is renamed.

Caution
Do not kill the `mboxutil` process in the middle of execution. If it is killed with `SIGKILL` (`kill -9`), it may potentially require that every server get restarted and a recovery be done.

Examples

To list all mailboxes for all users:

```
mboxutil -l
```

To list all mailboxes and also include path and ACL information:

```
mboxutil -l -x
```

To create the default mailbox named `INBOX` for the user `daphne`:

```
mboxutil -c user/daphne/INBOX
```

To delete a mail folder named `projx` for the user `delilah`:

```
mboxutil -d user/delilah/projx
```

To delete the default mailbox named `INBOX` and *all mail folders* for the user `druscilla`:

```
mboxutil -d user/druscilla/INBOX
```

To rename the mail folder `memos` to `memos-april` for the user `desdemona`:

```
mboxutil -r user/desdemona/memos user/desdemona/memos-april
```

To move the mail account for the user `dimitria` to a new partition:

```
mboxutil -r user/dimitria/INBOX user/dimitria/INBOX partition
```

where *partition* specifies the name of the new partition.
To move the mail folder named `personal` for the user `dimitria` to a new partition:

```
mboxutil -r user/dimitria/personal user/dimitria/personal partition
```

To Move Mailboxes to a Different Disk Partition

By default, mailboxes are created in the `primary` partition. If the partition gets full, additional messages cannot be stored. There are several ways to address the problem:

- Reduce the size of user mailboxes
- If you are using volume management software, add additional disks
- Create additional partitions (To Add a Message Store Partition) and move mailboxes to the new partitions

If possible, we recommend adding additional disk space to a system using volume management software since this procedure is the most transparent for the user. However, you may also move mailboxes to a different partition.

To move mailboxes to a different partition, see the preceding `mboxutil -r` examples.

To Remove Orphan Accounts

To search for orphaned accounts (orphaned accounts are mailboxes that do not have corresponding entries in LDAP) use the following `mboxutil` commands:

```
mboxutil -o
```

Command output follows:

```
mboxutil: Start checking for orphaned mailboxes
  user/annie/INBOX
  user/oliver/INBOX
  mboxutil: Found 2 orphaned mailbox(es)
  mboxutil: Done checking for orphaned mailboxes
```

Use the following command to create a file listing orphaned mailboxes that can be turned into a script file that deletes the orphaned mailboxes (example filename is `orphans.cmd`):

```
mboxutil -o -w orphans.cmd
```

The command output is as follows:

```
mboxutil: Start checking for orphaned mailboxes
  mboxutil: Found 2 orphaned mailbox(es)
  mboxutil: Done checking for orphaned mailboxes
```

Delete the orphan files with the following command:

```
mboxutil -d -f orphans.cmd
```

To Find a Mailbox's Directory Using hashdir

The mailboxes in the message store are stored in a hash structure for fast searching. Consequently, to find the directory that contains a particular user's mailbox, use the `hashdir` utility.

This utility identifies the directory that contains the message store for a particular account. This utility reports the relative path to the message store, such as `d1/a7/`. The path is relative to the directory level just before the one based on the user ID. The utility sends the path information to the standard output.

For example, to find the relative path to the mailbox for user `crowe`:

```
hashdir crowe
```

To Find Out How Many Users Have Read Messages in a Shared Folder

The `readership` utility reports on how many users other than the mailbox owner have read messages in a shared IMAP folder (see Message Store Shared Folders Overview).

An owner of a IMAP folder may grant permission for others to read mail in the folder. A folder that others are allowed to access is called a *shared folder*. Administrators can use the `readership` utility to see how many users other than the owner are accessing a shared folder.

This utility scans all mailboxes and produces one line of output per shared folder, reporting the number of readers followed by a space and the name of the mailbox.

Each reader is a distinct authentication identity that has selected the shared folder within the past specified number of days. Users are not counted as reading their own personal mailboxes. Personal mailboxes are not reported unless there is at least one reader other than the folder's owner.

For example, the following command counts as a reader any identity that has selected the shared IMAP folder within the last 15 days:

```
readership -d 15
```

Managing Message Store Partitions and Adding Storage

Managing Message Store Partitions and Adding Storage

This information describes message store partitions and adding storage.

Topics:

See also:

- To Move Mailboxes to a Different Disk Partition

Message Store Partition Overview

Mailboxes are stored in message store partitions, an area on a disk partition specifically devoted to storing the message store. Message store partitions are not the same as disk partitions, though for ease of maintenance, it is recommended that you have one disk partition and one file system for each message store partition. Message store partitions are directories specifically designated as a message store.

User mailboxes are stored by default in the *store_root*`/partition/` directory (see Message Store Directory Layout). The `partition` directory is a logical directory that might contain a single partition or multiple partitions. At start-up time, the `partition` directory contains one subpartition called the `primary` partition.

You can add partitions to the `partition` directory as necessary. For example, you might want to partition a single disk to organize your users as follows:

```
store_root/partition/mkting/
store_root/partition/eng/
store_root/partition/sales/
```

As disk storage requirements increase, you might want to map these partitions to different physical disk drives.

You should limit the number of mailboxes on any one disk. Distributing mailboxes across disks improves message delivery time (although it does not necessarily change the SMTP accept rate). The number of mailboxes you allocate per disk depends on the disk capacity and the amount of disk space allocated to each user. For example, you can allocate more mailboxes per disk if you allocate less disk space per user.

If your message store requires multiple disks, you can use RAID (Redundant Array of Inexpensive Disks) technology to ease management of multiple disks. With RAID technology, you can spread data across a series of disks but the disks appear as one logical volume so disk management is simplified. You might also want to use RAID technology for redundancy purposes; that is, to duplicate the store for failure recovery purposes.

Note
To improve disk access, the message store and the message queue should reside on separate disks.

To Add a Message Store Partition

When adding a partition, you specify both an absolute physical path where the partition is stored on disk, and a logical name (called the partition nickname).

The partition nickname allows you to map users to a logical partition name regardless of the physical path. When setting up user accounts and specifying the message store for a user, you can use the partition nickname. The name you enter must be an alphanumeric name and must use lowercase letters.

To create and manage the partition, the user ID used to run the server must have permission to write to the location specified in the physical path.

Note
After adding a partition, you must stop then restart the server to refresh the configuration information.

- **Command Line,** To add a partition to the store at the command line:
  `configutil -o store.partition.`*nickname*`.path -v` *path*
  where *nickname* is the logical name of the partition and *path* indicates the absolute path name where the partition is stored.
  To specify the path of the default primary partition:

```
configutil -o store.partition.primary.path -v path
```

To Change the Default Message Store Partition

The default partition is the partition used when a user is created and the `mailMessageStore` LDAP attribute is not specified in the user entry. The `mailMessageStore` LDAP attribute, which specifies a user's message store partition, should be specified in all user entries so that a default partition is not necessary. In addition, the default partition should **not** be changed for load balancing or any other reason. It is invalid and dangerous to change the default partition while there are still users depending on the default partition definition.

If it is absolutely necessary to change the default partition, make sure that all users on the old default partition (the one being left behind) have their `mailMessageStore` attribute set to their current partition (which will no longer be the default), before changing the definition of default with the `configutil` parameter `store.defaultpartition`.

Adding More Physical Disks to the Message Store

The Messaging Server message store contains the user mailboxes for a particular Messaging Server instance. The size of the message store increases as the number of mailboxes, folders, and log files increase.

As you add more users to your system, your disk storage requirements increase. Depending on the number of users your server supports, the message store might require one physical disk or multiple physical disks. Messaging Server enables you an add more stores as needed. See Using Sun

StorageTek 53xx NAS with Messaging Server Message Store or Using NetApp Filers with Sun Java System Messaging Server Message Store for information on how to add disks to the message store.

Managing Message Store Quotas

Managing Message Store Quotas

This information describes the quota tasks. See Message Store Quota (Overview) for conceptual information.

Topics:

See also Monitoring the Message Store.

To Specify a Default User Quota

A default quota applies to users who do not have individual quotas set in their LDAP entries. The process consists of two steps: 1) Specifying a user default quota and 2) Specifying which users are bound to the default quota. The following examples show how to set default user quotas. Refer to http://msg.wikidoc.info/index.php/Configutil_Reference for detailed parameter information.

- To specify a default user disk quota for message size in bytes:

```
configutil -o store.defaultmailboxquota -v [ -1 | <number> ]
```

where `-1` indicates no quota (unlimited message usage) and *number* indicates a number of bytes.

- To specify a default user quota for total number of messages:

```
configutil -o store.defaultmessagequota -v [ -1 | <number> ]
```

where `-1` indicates no quota (unlimited messages) and *number* indicates the number of messages.

- To specify the default quota for specific users, set the `mailQuota` attribute to `-2` in the user entries that use the default message store quota. Note that if `mailQuota` is not specified, the system default quota is used.

To Specify Individual User Quotas

Each user can have individualized quotas. To set user-specific quotas, set the `mailQuota` or `mailMsgQuota` attributes in the user's LDAP entry (see *Unified Communications Suite Schema Reference* for more information).

In addition, a number of `configutil` parameters of the form `store.quota*` can be used to implement more fine grained quota policies. See http://msg.wikidoc.info/index.php/Configutil_Reference. The

following examples show how to set user quotas.

To specify the system default quota, do not add `mailQuota` to the LDAP entry, or set it to `-2`.

To set the quota to 1,000 messages set `mailMsgQuota` to `1000`.

To set the quota to two megabytes set `mailQuota` to `2M`.

To set the quota to two gigabytes, set `mailQuota` to `2G` or `2000M`.

To specify a 2 Gigabyte quota; a 20 Megabyte voice mail quota; and a 100 Megabyte quota for the Archive folder:
`mailQuota: 2G;#voice%20M;Archive%100M`

The two gigabyte quota represents all folders in the user's mailbox that are not explicitly assigned quotas. In this example, that excludes messages in the `Archive` folder, and messages of type `voice`. The 100 Megabyte quota includes messages in any folders within the `Archive` folder.

Refer to [`mailQuota` and "Specifying Quotas for Folders and Message Types" in *Unified Communications Suite Schema Reference* for more information. See also Managing Message Types in the Message Store.

To Specify Domain Quotas

You can set disk space or message quotas for domains. These quotas are for the cumulative bytes or messages of all users in a particular domain. To specify domain quotas, set the `mailDomainDiskQuota` or `mailDomainMsgQuota` attributes.

To set the quota to 1,000 messages set `mailDomainMsgQuota` to `1000`.

To set the quota to two megabytes set `mailDomainDiskQuota` to `2M` or `2000000`.

To set the quota to two gigabytes, set `mailDomainDiskQuota` to `2G` or `2000000000` or `2000M`.

To Specify a Quota on a Subfolder that is Different from the Parent Folder

You can set a quota on a subfolder that is different from the quota on its parent folder.

1. Make sure that folder quota is enabled.

   ```
   configutil -o store.folderquota.enable -v 1
   ```

2. Stop and restart the IMAP service to put the folder quota enablement into effect.
3. Update the user's `mailQuota` attribute to set the quota on the particular folder.
   For example, setting the `mailQuota` attribute as follows specifies 20 Mbytes for the user's entire mailbox tree, and 30 Mbytes for the folder named `Archive`.

   ```
   mailQuota:20M;Archive%30M
   ```

4. Confirm the value of the `mailQuota` attribute.

```
# ldapsearch -D cn=directory manager -w password -b o=isp uid=kellyc
mailquota
version: 1
dn: uid=kellyc,ou=People,o=east.example.com,o=isp
mailquota: 20M;Archive%30M
```

5. Obtain the quota status of the user.

```
# ./imquotacheck -u kellyc
Name Quota(K) Usage(K) % Quota# Usage# % OverDate WarnDate
-------------------- -------- -------- --- ------- ------- --- --------
--------
kellyc - 20327 - - 406 - - -
-------------------------------------------------------------------------
```

6. Reinitialize the quota limit from the LDAP directory and recalculate the total amount of disk space that is being used by the user.

```
# ./iminitquota -u kellyc
kellyc: updated
```

7. Confirm that the quota status has been updated.

```
# ./imquotacheck -u kellyc
Name Quota(K) Usage(K) % Quota# Usage# % OverDate WarnDate
-------------------- -------- -------- --- ------- ------- --- --------
--------
kellyc 20480 20327 99 - 406 - 12/20/12 -
kellyc/Archive 30720 0 0 - 0 - - -
-------------------------------------------------------------------------
```

This example now shows two quotas: the first on the entire mailbox, and the second for the folder `Archive`. Note that the `Archive` folder does not yet exist.

8. Have the user create the `Archive` folder.
9. Once the user starts using the folder and moving messages into it, you can check the status.

```
# ./imquotacheck -u kellyc
Name Quota(K) Usage(K) % Quota# Usage# % OverDate WarnDate
-------------------- -------- -------- --- ------- ------- --- --------
--------
kellyc 20480 20327 99 - 406 - 12/20/12 -
kellyc/Archive 30720 57 0 - 2 - - -
-------------------------------------------------------------------------
```

To Set Up Quota Notification

Quota notification is the process of sending users a warning message when they are getting close to

their quota. Using this feature requires three steps.

1. **Enable Quota Notification**

   Run the following at the command line:

   ```
   configutil -o store.quotanotification -v [ yes | no ]
   ```

   If the message is not set, no quota warning message is sent to the user.

2. **Define a Quota Warning Message**

   The Warning Message is the message that will be sent to users who are close to exceeding their disk quota. To define a quota warning message at the command line:

   `configutil -o store.quotaexceededmsg -v` '*message*'

   The message must be in RFC 822 format. It must contain a header with at least a subject line, follow by $$, then the message body. ";" represents a new line. Depending on the shell that you are using, it might be necessary to append a \ before $ to escape the special meaning of $. ($ is often the escape character for the shell.) Example:

   ```
   configutil -o store.quotaexceededmsg -v 'Subject: WARNING: User quota
   exceeded$$User quota threshold exceeded - reduce space used.'
   ```

   In addition, there is support for the following variables:

   `[ID]` - userid

   `[DISKUSAGE]` - disk usage

   `[NUMMSG]` - number of messages

   `[PERCENT]` - `store.quotawarn` percentage

   `[QUOTA]` - `mailquota` attribute

   `[MSGQUOTA]` - `mailmsgquota` attribute

   Here's an example, using these variables:

   ```
   configutil -o store.quotaexceededmsg -v 'Subject: Overquota
   Warning$$[ID],$$Your mailbox size has exceeded [PERCENT] of its alloted
   quota.$Disk Usage: [DISKUSAGE]$Number of Messages: [NUMMSG]$Mailquota:
   [QUOTA]$Message Quota: [MSGQUOTA]$$-Postmaster'
   ```

3. **Specify how often the warning message is sent.**

   Set the following parameter:

   `configutil -o store.quotaexceededmsginterval -v` *number*

   where *number* indicates a number of days. For example, 3 would mean the message is sent every 3 days.

4. **Specify a Quota Threshold**

A quota threshold is a percentage of a quota that is exceeded before clients are sent a warning. When a user's disk usage exceeds the specified threshold, the server sends a warning message to the user.

> **Note**
> When `local.store.quotaoverdraft=on` email notifications are not triggered until the user's disk usage exceeds 100% of the quota regardless of the threshold set with `store.quotawarn`.

For IMAP users whose clients support the IMAP ALERT mechanism, the message is displayed on the user's screen each time the user selects a mailbox and a message is also written to the IMAP log.

To specify a quota threshold at the command line:

`configutil -o store.quotawarn -v` *number*

where *number* indicates a percentage of the allowed quota.

To Disable Quota Notification

To disable quota notification, set `store.quotanotification` to no.

```
configutil -o store.quotanotification -v no
```

To Enable or Disable Quota Enforcement

By default, users or domains can exceed their quotas with no effect except for receiving an over quota notification (if set). Quota enforcement locks the mailboxes from receiving further messages until the disk usage is reduced below the quota level.

To Enable Quota Enforcement at the User level

```
configutil -o store.quotaenforcement -v [ on | off]
```

The MTA saves over-quota messages in its queues and notifies users that their messages wer not delivered but that a redelivery attempt is to be made later. Delivery retries continue until either the grace period expires and all messages are sent back to the senders, or the disk usage falls below the quota and messages can be dequeued from the MTA and delivered to the message store. If you want to return messages that are over quota before they get to the message queues, use the following command:

```
configutil -o local.store.overquotastatus -v on
```

To Perform Quota Enforcement at the Domain Level

Unlike user-level quotas, domain-level quotas are not maintained dynamically. To enforce quotas for a particular domain, use the following command:

```
imquotacheck -f -d <domain>
```

To enforce quotas for all domains, exclude the `-d` option. When `imquotacheck -f` finds a domain with `maildomainstatus=active` that has exceeded its quota, the `maildomainstatus` attribute is set to `overquota`, which halts all delivery to this domain. When `imquotacheck -f` is run again and the domain is back under quota, the value is set to `active`.

Disabling Quota Enforcement

If it appears that user quotas are being enforced, even when you have disabled them, check that the following `configutil` parameters are off or not set:

- `store.quotaenforcement`
- `local.store.overquotastatus`
- `local.store.quotaoverdraft`

When `local.store.overquotastatus` is on, it always treats `store.quotaoverdraft` as on, otherwise users never go over quota to trigger the rejection. Also, when `store.quotaoverdraft` is on , users are allowed one message that is smaller than the quota only. That is, it never accepts a message that is greater than the user's quota.

After changing these parameters, restart your messaging services.

These Message Store attributes should be active:

- `maildomainstatus`
- `mailuserstatus`

Messages bounce if they are larger than the mailbox quota, regardless of quota enforcement configuration.

To Set a Grace Period

The grace period specifies how long the mailbox can be over the quota (disk space or number of messages) before messages are bounced back to sender. The grace period is not how long the message is held in the message queue, it's how long the mailbox is over quota before all incoming messages, including those in the message queue, are bounced. (see Message Store Quota (Overview) for more details.) The grace period starts when the user has reached the quota threshold and been warned. See To Setup Quota Notification.

To specify a quota grace period at the command line:

```
configutil -o store.quotagraceperiod -v number
```

where *number* indicates number of hours.

Netscape Messaging Server Quota Compatibility Mode

After disk usage exceeded the quota in the Netscape Messaging Server, the server deferred or bounced message delivery, sent an over quota notification, and started the grace period. Messaging Server provides a parameter, `local.store.quotaoverdraft`, which retains this behavior.

When set to `ON`, messages are delivered until disk usage is over quota. At that time, messages are deferred (messages stay in the MTA message queue but are not delivered to the message store), an over quota warning message is sent to the user, and a grace period starts. The grace period determines

how long a mailbox is overquota before the overquota messages bounce. (The default is that the quota warning messages are sent when the message store reaches the threshold.) The default for this parameter is `Off`.

Managing Message Types in the Message Store

Managing Message Types in the Message Store

This information describes how to work with message types. For conceptual information, see Message Store Message Types Overview.

Topics:

To Configure Message Types

To configure a message type, use the `configutil` utility to set the `store.messagetype` parameter values that define and identify the message type.

1. Enable message types by setting the `store.messagetype.enable` parameter to on.
   This `configutil` parameter allows the message store to identify and manipulate message types. You must set this parameter before you can configure an individual message type. For example, enter the following command:

   ```
   configutil -o store.messagetype.enable -v 1
   ```

2. Define and identify the message type by setting the `store.messagetype.`*x* parameter.
   The variable *x* identifies this particular message type in the message store. The variable *x* must be an integer greater than zero and less than 64. You can define up to 63 message types by iteratively configuring this parameter with unique integers. You define the value of the message type with a text string that describes the type.
   - For example, to define a text message type, type the following command:

     ```
     configutil -o store.messagetype.1 -v text/plain
     ```

   - To define a voice message type, type the following command:

     ```
     configutil -o store.messagetype.2 -v multipart/voice-message
     ```

3. Provide a flag name for the message type by setting the `store.messagetype.`*x*`.flagname` parameter.
   This parameter creates a unique flag that identifies the message type. The flag is automatically set whenever a message of this type first arrives in the message store and remains associated with the message until it is purged. The flag name value is a text string that describes the message type. It does not have to be the same as the value set with the `store.messagetype.`*x* parameter. The variable *x* is the integer ID of the message type defined with the `store.messagetype.`*x* parameter. For example, to define flag names for the message types configured in the preceding step, type the following commands:

```
configutil -o store.messagetype.1.flagname -v text
configutil -o store.messagetype.2.flagname -v voice_message
```

4. Configure a quota root name for the message type by setting the `store.messagetype.`*x*`.quotaroot` parameter.
   This parameter enables the quota function to identify and manage a quota root for this message type. The parameter value is a name---a text string that describes the message type. It does not have to be the same as the value set with the `store.messagetype.`*x* parameter. The variable *x* is the integer ID of the message type defined with the `store.messagetype.`*x* parameter. When this parameter is configured, you can set a quota that applies to the specified message type. For more information, see Administering Quotas by Message Type.

   For example, to enable the use of quota roots for the message types configured in the preceding steps, type the following commands:

   ```
   configutil -o store.messagetype.1.quotaroot -v text
   configutil -o store.messagetype.2.quotaroot -v voice
   ```

5. To configure an alternate header field for identifying the message type, set the `store.messagetype.header` parameter.

   By default, the message store reads the Content-Type header field to determine the message type. Configure the `store.messagetype.header` parameter only if you want to use a different header field for identifying the message type. The value of this parameter is a text string.

   For example, to use a field called `Message-Context`, enter the following command:

   ```
   configutil -o store.messagetype.header -v Message-Context
   ```

Sending Notification Messages for Message Types

Notifications can deliver status information about messages of different types, such as text messages, voice mail, and image data. Messaging Server uses Message Queue to send notification information for message types. For information about configuring the JMQ notification plug-in for Message Queue, see Configuring a JMQ Notification Service (Task).

To enable the JMQ notification plug-in to recognize a particular message type, you must configure the `store.messagetype` parameters, including the `store.messagetype.`*x*`.flagname` parameter. For details, see To Configure Message Types.

Once the message types have been configured, JMQ notification messages can identify the particular message types. You can write a Message Queue client to interpret notification messages by message type and deliver status information about each type to the mail client.

The JMQ notification function counts the number of messages currently in the mailbox, by message type. Instead of sending one count, an array specifying the count for each message type is sent with the notification message.

For example, a `NewMsg` notification message can carry data to tell the user that, for example, there are seven new voice mail messages and four new text messages in the user's inbox.

For more information about sending notifications by message type, see Notifications for Particular Message Types.

Administering Quotas by Message Type

When you set a quota for a message type, you include that value in a *quota root*. A quota root specifies quotas for a user. It can specify different quotas for particular message types and mailbox folders, and it can specify a default quota that applies to all remaining message types, folders, and messages not defined by type.

For complete information about setting and managing quotas, see Quota Theory of Operations

Before You Set Message-Type Quotas

Before you can set quotas for message types, you must configure the following parameters:

- Set the `store.messagetype.x.quotaroot` parameter for each message type. For details, see To Configure Message Types.
- Set the `store.typequota.enable` parameter to `on`.

  For example, enter the following command:

```
configutil -o store.typequota.enable -v 1
```

Methods of Setting Message-Type Quotas

Use one of the following methods to set quotas for message types:

- Set message-type quotas for a user with the LDAP attributes `mailQuota` or `mailMsgQuota` (or both).

  For information about how to set quota roots with these attributes, see the `mailQuota` and `mailMsgQuota` entries in *Unified Communications Suite Schema Reference*.
- Set default message-type quotas that apply to all individual users when the `mailQuota` and `mailMsgQuota` attributes are not set.

  To set default quotas, use the `store.defaultmessagequota` or `store.defaultmailboxquota` parameter (or both).

  For information about how to set quota roots with these parameters, see Managing Message Store Quotas.

When you set a quota for the message type with a `configutil` parameter or LDAP attribute shown above, you must use the quota root specified with the `store.messagetype.x.quotaroot` parameter.

Example of a Message-Type Quota Root

The example described in this section sets the following quotas for the user `joe`:

- The default mailbox storage quota is 40 M
- The default mailbox message quota is 5000
- The storage quota for the Archive folder is 100M
- The storage quota for text message types is 10 M
- The message quota for text message types is 2000
- The storage quota for voice message types is 10 M
- The message quota for voice message types is 200

This quota root permits greater storage in the Archive folder (100 M) than in all the other folders and message types combined (60 M). Also, no message limit is set for the Archive folder; in this example, only storage limits matter for archiving.

The message types have both storage and number-of-message quotas.

The message-type quotas apply to the sum of all messages of those types, whether they are stored in the Archive folder or in any other folder.

The default mailbox quotas apply to all messages that are not text or voice message types and are not stored in the Archive folder. That is, the message-type quotas and Archive quota are not counted as part of the default mailbox quotas.

To set the quota root in this example, you would take the following steps:

1. Configure the `store.messagetype.`*`x`*`.quotaroot` parameter as follows:

```
store.messagetype.1.quotaroot = text
store.messagetype.2.quotaroot = voice
```

2. Configure the `mailQuota` attribute for the user `joe` as follows:

```
mailQuota: 20M;#text%10M;#voice%10M;Archive%100M
```

3. Configure the `mailMsgQuota` attribute for the user `joe` as follows:

```
mailMsgQuota: 5000;#text%2000;#voice%200
```

When you run the `getquotaroot` IMAP command, the resulting IMAP session displays all quota roots for the user `joe`'s mailbox, as shown here:

```
1 getquotaroot INBOX
* QUOTAROOT INBOX user/joe user/joe/#text user/joe/#voice
* QUOTA user/joe (STORAGE 12340 20480 MESSAGE 148 5000)
* QUOTA user/joe/#text (STORAGE 1966 10240 MESSAGE 92 2000)
* QUOTA user/joe/#voice (STORAGE 7050 10240 MESSAGE 24 200)
2 getquotaroot Archive
* QUOTAROOT user/joe/Archive user/joe/#text user/joe/#voice
* QUOTA user/joe/Archive (STORAGE 35424 102400)
* QUOTA user/joe/#text (STORAGE 1966 10240 MESSAGE 92 2000)
* QUOTA user/joe/#voice (STORAGE 7050 10240 MESSAGE 24 200)
```

Expiring Messages by Message Type

The expire and purge feature allows you to move messages from one folder to another, archive messages, and remove messages from the message store, according to criteria you define in expire rules. You perform these tasks with the `imexpire` utility.

Because the `imexpire` utility is run by the administrator, it bypasses quota enforcement.

For information about how to write expire rules and use the `imexpire` utility, see Configuring Message Expiration.

You can write expire rules so that messages of different types are expired according to different criteria.

The expire feature is extremely flexible, offering many choices for setting expire criteria. This section describes one example in which text and voice messages are expired according to different criteria.

The example assumes you have configured text and voice message types as follows:

```
store.messagetype.1 = text/plain
store.messagetype.2 = multipart/voice-message
```

Assume also that the message store is configured to read the Content-Type header field to determine the message type.

Example: Sample Rules for Expiring Different Message Types

```
TextInbox.folderpattern: user/%/INBOX
TextInbox.messageheader.Content-Type: text/plain
TextInbox.messagedays: 365
TextInbox.action: fileinto:Archive
VoiceInbox.folderpattern: user/%/INBOX
VoiceInbox.messageheader.Content-Type: multipart/voice-message
VoiceInbox.savedays: 14
VoiceInbox.action: fileinto:OldMail
VoiceOldMail.folderpattern: user/%/OldMail
VoiceOldMail.messageheader.Content-Type: multipart/voice-message
VoiceOldMail.savedays: 30
VoiceOldMail.action: fileinto:Trash
Trash.folderpattern: user/%/Trash
Trash.savedays: 7
Trash.action: discard
```

In this example, text messages and voice mail are expired in different ways, and they follow different schedules, as follows:

- Text messages are moved from a user's inbox to the user's Archive folder one year after they arrive in the message store.
- Voice mail is moved from the inbox to the OldMail folder after two weeks. If the user saves a voice message, the saved date is reset, and the message is moved two weeks after the new date.
- Voice mail is moved from the OldMail folder to the Trash folder after 30 days. The user also can save a voice message in the OldMail folder, which postpones the removal of the message for another 30 days after the new saved date.
- Messages of all types are discarded seven days after they are moved to the Trash folder.

  The expire rules move voice mail to Trash automatically. Text messages are moved to Trash when a user deletes them.

Note: The `savedays` rule causes a message to be expired the specified number of days after the message is saved. In a typical voice mail system, a user can save voice mail on the voice mail menu. For text messages, a message is saved when it is moved to a folder. The `messagedays` rule causes a message to be expired the specified number of days after it first arrives in the message store, no matter which folder it is stored in or how often it is moved.

Managing Shared Folders

Shared Folder Tasks

This information describes the tasks that you use to administer shared folders. See Message Store Shared Folders Overview for conceptual information.

Topics:

To Specify Sharing Attributes for Private Shared Folders

A private shared folder is a normal folder, created by users in the same way that they create other folders. A folder becomes "shared" when its owner grants access rights to other users or groups. Methods to manage folder access include:

- Many IMAP clients
- Communications Express web client
- Messaging Server `readership` command, for mail administrators

The following table explains the `configutil` parameters that pertain to private shared folders:

`configutil` Parameter	Description	Default
`store.privatesharedfolders.restrictanyone`	If enabled (1), disallows regular users from setting the permission on private shared folders to `anyone` .	0
`store.privatesharedfolders.restrictdomain`	If enabled (1), disallows regular users sharing private folders to users outside of their domain.	0
`store.privatesharedfolders.shareflags`	If disabled (0), users of a shared folder have their own set of flags (for example, *seen*, *deleted*, and so on) for messages in that folder. If enabled (1), a single set of flags is shared between all users of each shared folder.	0

To Create a Public Shared Folder

Public shared folders must be created by the mail administrator because they require access to the LDAP database as well as the `readership` and `mboxutil` commands.

1. Set the userid for Public shared folders.

The `store.publicsharedfolders.user` configutil option specifies the userid to act as a container for all public shared folders (see Message Store Shared Folders Overview). Typically, this is simply `public`. The default is NULL (unset).

```
configutil -o store.publicsharedfolders.user -v public
```

2. Create an LDAP entry for that user. The `uid` must match that specified by `store.publicsharedfolders.user`, for example:

```
dn: cn=public,ou=people,o=sesta.com,o=ISP
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: inetUser
objectClass: ipUser
objectClass: inetMailUser
objectClass: inetLocalMailRecipient
objectClass: nsManagedPerson
objectClass: userPresenceProfile
cn: public
mail: public@sesta.com
mailDeliveryOption: mailbox
mailHost: manatee.siroe.com
uid: public
inetUserStatus: active
mailUserStatus: active
mailQuota: -1
mailMsgQuota: 100
```

3. Create folders within the `public` account using the `mboxutil` command, for example:

```
mboxutil -c user/public/gardening
```

4. Use the `readership` command grant rights to allow users to access the folder.
   For example the following command gives everyone in the `sesta.com` domain lookup, read, and posting access to the public folder `gardening`:

```
readership -s user/public/gardening anyone@sesta.com lrp
```

   The name `anyone@<domain>` is a special case to designate all users in the specified domain. It does not correspond to any user or group definition in LDAP. The name `anyone` without specifying a domain indicates anyone in any domain.
   To grant the user whose `uid` is `kelly` the same access rights as the owner of the folder:

```
readership -s user/public/gardening kelly@sesta.com lrswipcdan
```

   For individual users, you only need to supply a domain name with hosted domains. Do not use a domain name if the user to whom access is being granted is in the default domain.
   The `readership` command documentation lists the ACL Rights Characters and their meanings.

To Grant Folder Access Rights Based on Group Membership

In the previous examples, access rights have been granted to individual users or to the special case names `anyone` or `anyone@<domain>`. You can also grant rights based on group membership. Members of such a group are identified by having the `aclGroupAddr` attribute.

For example, a group called `tennis@sesta.com` has 25 members and the members have decided that they would like to create a shared folder to store all email going to this group address and to allow members of the group to access that shared folder.

The mail administrator uses the `readership` command to grant group access rights. A group name is distinguished from individual user names by the prefix "`group=`".

1. Create the folder.
   In this example, the team decided to use a private shared folder. The user `gregk` could have created the folder using his mail client, or the mail administrator could have created it with `mboxutil`, for example:

   ```
   mboxutil -c user/gregk/gardening
   ```

   Or if they were using a public shared folder, the mail administrator would have had to create it:

   ```
   mboxutil -c user/public/gardening
   ```

2. Use the `readership` command to grant *lookup*, *read*, and *posting* access privileges to the group:

   ```
   readership -s user/gregk/gardening group=tennis@sesta.com lrp
   ```

3. Assign group membership to the individual users.
   For the purpose of folder access control, group membership is determined by the `aclGroupAddr` attribute on the LDAP entry of the individual users. Add the attribute-value pair `aclGroupAddr=<group-name>` to the user entry of every member of the group, for example:

   ```
   aclGroupAddr: tennis@sesta.com
   ```

To create group objects in LDAP, you could use the `aclGroupAddr` attribute as the basis for a dynamic group, for example:

```
memberURL:
ldap:///o=sesta.com??sub?(&(aclGroupAddr=tennis@sesta.com)(objectclass=inetmail
```

However, note that the LDAP group object with mail address `tennis@sesta.com` is not used for determining group membership for the purpose of shared folder access. What matters is that the "`xxx`" value in `group=xxx` on the `readership` command matches the value of the `aclGroupAddr` attribute on the user's LDAP object.

Also note that if you use the `aclGroupAddr` attribute as the criteria for a dynamic group, you should check to make sure that attribute is indexed properly for such lookups.

To Set or Change a Shared Folder's Access Control Rights

Users can set or change the access control for a shared folder using the Communications Express interface. Administrators can set or change the access control for a shared folder using the `readership` command line utility. The command has the following form:

```
readership -s <foldername> <identifier> <rights_chars>
```

where *foldername* is the name of the folder for which you are setting rights, *identifier* is the person or group to whom you are assigning the rights, and *rights_chars* are the rights you are assigning. For the meaning of each character, see the table below.

> **Note**
> `anyone` is a special identifier. The access rights for `anyone` apply to all users. Similarly, the access rights for `anyone@`*domain* apply to all users in the same domain. For the identifier, only supply a domain name with hosted domains. Do not use a domain name if the folder is in the default domain.

Examples

If you wish everyone at the `sesta` domain to have lookup, read and email marking (but not posting) access to the public folder called `golftournament`, issue the command as follows:

```
readership -s user/public/golftournament anyone@sesta lwr
```

To assign the same access to everyone on the message store issue the following:

```
readership -s user/public/golftournament anyone lwr
```

To assign lookup, read, email marking and posting rights to a group, issue the command as follows:

```
readership -s user/public/golftournament group=golf@sesta.com lwrp
```

If you want to assign administrator and posting rights for this folder to an individual, `jdoe`, issue the command as follows:

```
readership -s user/public/golftournament jdoe@sesta.com lwrpa
```

To deny an individual or group access to a public folder, prefix the `userid` with a dash. For example, to deny lookup, read and write rights to `jsmith`, issue the command as follows:

```
readership -s user/public/golftournament -jsmith@sesta.com lwr
```

To deny an individual or group an access right, prefix the ACL rights character with a dash. For example, to deny posting rights to `jsmith`, issue the command as follows:

```
readership -s user/public/golftournament jsmith@sesta.com -p
```

To remove an individual or group access right setting from a folder, set it to an empty set. This is different from an ACL to deny access:

```
readership -s user/public/golftournament jsmith@sesta.com ""
```

Note
Posting messages to a shared folder using the *uid+folder@domain* address requires that the p (post) access right be used with the readership command. See To Set or Change a Shared Folder's Access Control Rights.

To Enable or Disable Listing of Shared Folders

The configuration option `local.store.sharedfolders` allows you to enable to disable listing of shared folders when responding to an IMAP `LIST` command. Setting the option to `off` disables it. The setting is enabled by default (set to `on`).

`SELECT` and `LSUB` commands are not affected by this option. The `LSUB` command returns every subscribed folder, including shared folders. Users can `SELECT` the shared folders they own or are subscribed to.

To Set Up Distributed Shared Folders

Normally, shared folders are only available to users on a particular message store. Messaging Server, however, allows you to create *distributed shared folders* that can be accessed across multiple message stores. That is, access rights to distributed shared folders can be granted to any users within the group of message stores.

Distributed shared folders require the following:

- Every message store `userid` must be unique across the group of message stores.
- The directory data across the deployment must be identical.

The remote message stores (that is the message stores that do not hold the shared folder) must be configured as proxy servers by setting the configuration variables listed in the following table.

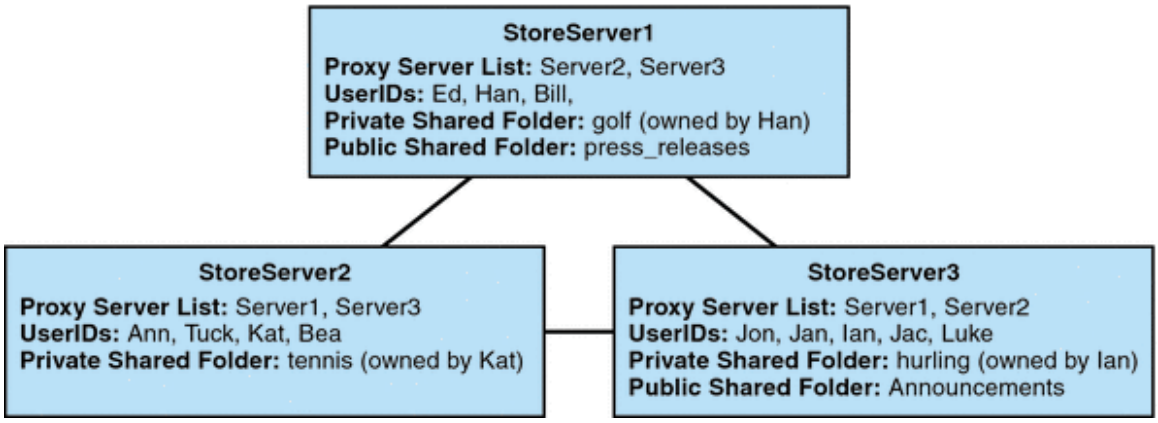
Variables for Configuring Distributed Shared Folders

Name	Value	Data Format
`local.service.proxy.serverlist`	message store server list	space-separated strings
`local.service.proxy.admin`	default store admin login name	string
`local.service.proxy.adminpass`	default store admin password	string
`local.service.proxy.admin.`*hostname*	store admin login name for a specific host	string
`local.service.proxy.adminpass.`*hostname*	store admin password for a specific host	string

Setting Up Distributed Shared Folders—Example

The following figure shows a distributed folder example of three message store servers called StoreServer1, StoreServer2, and StoreServer3.

Distributed Shared Folders—Example



These servers are connected to each other as peer proxy message stores by setting the appropriate `configutil` variables. Each server has a private shared folder—*golf* (owned by Han), *tennis* (owned by Kat), and *hurling* (owned by Luke). In addition there are two public shared folders called *press_releases* and *Announcements*. Users on any of the three servers can access any of these three shared folders.

The following example shows the ACLs for each server in this configuration.

```
$ StoreServer1 :> imcheck -d lright.db
Ed: user/Han/golf
Ian: user/Han/golf
anyone: user/public/press_releases
```

```
$ StoreServer2 :> imcheck -d lright.db
Jan: user/Kat/tennis
Ann: user/Kat/tennis
anyone: user/public+Announcements user/public+press_releases
```

```
$ StoreServer3 :> imcheck -d lright.db
Tuck: user/Ian/hurling
Ed: user/Ian/hurling
Jac: user/Ian/hurling
anyone: user/public/Announcements
```

To Monitor and Maintain Shared Folder Data

The `readership` command line utility allows you to monitor and maintain shared folder data which is held in the `folder.db`, `peruser.db`, and `lright.db` files. `folder.db` has a record for each folder that holds a copy of the ACLs. The `peruser.db` has an entry per user and mailbox that lists the various flags settings and the last date the user accessed any folders. The `lright.db` has a list of all the users and the shared folders for which they have lookup rights.

The `readership` command line utility takes the following options:

`readership` Options

Options	Description
`-d days`	Returns a report, per shared folder, of the number of users who have selected the folder within the specified days.
`-p months`	Removes data from the `peruser.db` for those users who have not selected their shared folders within the specified months.
`-l`	List the data in `lright.db.`
`-s` *folder_identifier_rights*	Set access rights for the specified folder. This updates the `lright.db` as well as the `folder.db`.

Using the various options, you can perform the following functions:

- To Monitor Shared Folder Usage
- To List Users and Their Shared Folders
- To Remove Inactive Users
- To Set Access Rights

To Monitor Shared Folder Usage

To find out how many users are actively accessing shared folders, issue the command:

```
readership -d days
```

where *days* is the number of days to check. Note that this option returns the number of active users, not a list of the active users.

Example: To find out the number of users who have selected shared folders within the last 30 days, issue the following command:

```
readership -d 30
```

To List Users and Their Shared Folders

To list users and the shared folders to which they have access, issue the command:

```
imcheck -d lright.db
```

Example output:

```
$ imcheck -d lright.db
group=lee-staff@siroe.com: user/user2/lee-staff
richb: user/golf user/user10/Drafts user/user2/lee-staff user/user10/Trash
han1: user/public+hurling@siroe.com user/golf
gregk: user/public+hurling@siroe.com user/heaving user/tennis
```

To Remove Inactive Users

If you want to remove inactive users (those who have not accessed shared folders in a specified time period) issue the command:

```
readership -p months
```

where *months* is the number of months to check for.

Example: Remove users who have not accessed shared folders for the past six months:

```
readership -p 6
```

To Set Access Rights

You can assign access rights to a new public folder, or change access rights on a current public folder.

For an example of how to set access rights with this command, see To Set or Change a Shared Folder's Access Control Rights.

Message Store Administration

Message Store Administration

The following lists all the administering the message store documentation. See also the main message store page.

Page: moveuser
Page: Protecting Mailboxes from Deletion or Renaming (Unified Configuration)
Page: Significant Changes in the Message Store Between Versions
Page: Administering Very Large Mailboxes in Unified Configuration
Page: Backing Up and Restoring the Message Store in Unified Configuration
Page: Managing Message Store Partitions and Adding Storage in Unified Configuration
Page: Administering Message Store Database Snapshots (Backups)
Page: Administering Very Large Mailboxes
Page: Backing Up and Restoring the Message Store
Page: Configuring POP, IMAP, and HTTP Services
Page: Managing Mailboxes
Page: Managing Message Store Partitions and Adding Storage
Page: Managing Message Store Quotas
Page: Message Store Command-line Utilities
Page: Message Store Maintenance Queue
Page: Protecting Mailboxes from Deletion or Renaming
Page: Specifying Administrator Access to the Message Store
Page: Migrating Mailboxes to a New System
Page: Managing Message Store Quotas in Unified Configuration
Page: Message Store and Mailbox Management in Unified Configuration
Page: Message Store Management

Message Store Architecture and Concepts

Message Store Architecture and Concepts

Message Store Automatic Recovery on Startup

Message Store Automatic Recovery On Startup

This information provides a conceptual overview of the startup and automatic recovery process of `stored`, and Message Store Database Snapshot. See Administering Message Store Database Snapshots (Backups) for task information.

Topics:

Overview of Automatic Recovery on Startup

Message store data consists of the messages, index data, and the message store database. While this data is fairly robust, on rare occasions there may be message store data problems in the system. These problems will be indicated in the default log file, and almost always will be fixed transparently. In rare cases an error message in the log file may indicate that you need to run the `reconstruct` utility. In addition, as a last resort, messages are protected by the backup and restore processes described in Backing Up and Restoring the Message Store.

The message store automates many recovery operations which were previously the responsibility of the administrator. These operations are performed by message store daemon `stored` during startup and include database snapshots and automatic fast recovery as necessary. `stored` thoroughly checks the message store's database and automatically initiates repairs if it detects a problem.

`stored` also provides a comprehensive analysis of the state of the database via status messages to the default log, reporting on repairs done to the message store and automatic attempts to bring it into operation.

Automatic Startup and Recovery Theory of Operations

The `stored` daemon starts before the other message store processes. It initializes and, if necessary, recovers the message store database. The message store database keeps folder, quota, subscription, and message flag information. The database is logging and transactional, so recovery is already built in. In addition, some database information is copied redundantly in the message index area for each folder.

Although the database is fairly robust, on the rare occasions that it breaks, in most cases `stored` recovers and repairs it transparently. However, whenever `stored` is restarted, you should check the default log files to make sure that additional administrative intervention is not required. Status messages in the log file will tell you to run `reconstruct` if the database requires further rebuilding.

Before opening the message store database, `stored` analyzes its integrity and sends status messages to the default log under the category of *warning*. Some messages will be useful to administrators and some messages will consist of coded data to be used for internal analysis. If `stored` detects any problems, it will attempt to fix the database and try starting it again.

When the database is opened, `stored` will signal that the rest of the services may start. If the automatic fixes failed, messages in the default log will specify what actions to take. See Error Messages Signifying that reconstruct Is Needed.

In previous releases, `stored` could start a recovery process which would take a very long time leaving the administrator wondering if `stored` was "stuck." This type of long recovery has been removed and

stored should determine a final state in less than a minute. However, if stored needs to employ recovery techniques such as recovering from a snapshot, the process may take a few minutes.

After most recoveries, the database will usually be up-to-date and nothing else will be required. However, some recoveries will require a reconstruct -m in order to synchronize redundant data in the message store. Again, this will be stated in the default log, so it is important to monitor the default log after a startup. Even though the message store will seem to be up and running normally, it is important to run any requested operations such as reconstruct.

Another reason for reading the log file is to determine what caused damage to the database in the first place. Although stored is designed to bring up the message store regardless of any problem on the system, you will still want to try to ascertain cause of the database damage as this may be a sign of a larger hidden problem.

Error Messages Signifying that reconstruct Is Needed

This section describes the type of error messages that require reconstruct to be run.

When the error message indicates mailbox error, run reconstruct <mailbox>. Example:

```
"Invalid cache data for msg 102 in mailbox user/joe/INBOX. Needs reconstruct"

"Mailbox corrupted, missing fixed headers: user/joe/INBOX"

"Mailbox corrupted, start_offset beyond EOF: user/joe/INBOX"
```

When the error message indicates a database error, run reconstruct -m. Example:

```
"Removing extra database logs. Run reconstruct -m soon after startup to
resync redundant data"

"Recovering database from snapshot. Run reconstruct -m soon after startup to
resync redundant data"
```

Message Store Database Snapshot Theory of Operations

This section describes concepts about the message store database snapshot. See Administering Message Store Database Snapshots (Backups) for a description of related tasks.

A snapshot is a hot backup of the database and is used by stored to restore a broken database transparently in a few minutes. This is much quicker than using reconstruct, which relies on the redundant information stored in other areas.

Snapshots of the database, located in the mboxlist directory, are taken automatically, by default, once every 24 hours. Snapshots are copied by default into a subdirectory of the store directory. By default, there are five snapshots kept at any given time: one live database, three snapshots, and one database/removed copy. The database/removed copy is newer and is an emergency copy of the database which is put into a subdirectory removed of the mboxlist database directory.

If the recovery process decides to remove the current database because it is determined to be bad, stored will move it into the removed directory if it can. This allows the database to be analyzed if desired.

The data move will only happen once a week. If there is already a copy of the database there, stored will not replace it every time the store comes up. It will only replace it if the data in the removed directory is older than a week. This will prevent the original database which had the problem from being replaced too soon by successive startups.

Message Store Database Snapshot Interval and Location

There should be five times as much space for the database and snapshots combined. It is highly recommended that the administrator reconfigure snapshots to run on a separate disk, and that it is tuned to the system's needs.

If `stored` detects a problem with the database on startup, the best snapshot will automatically be recovered. Three snapshot variables can set the following parameters: the location of the snapshot file, the interval for taking snapshots, number of snapshots saved. These `configutil` parameters are shown in Message Store Database Snapshot Parameters.

Having a snapshot interval which is too small will result in a frequent burden to the system and a greater chance that a problem in the database will be copied as a snapshot. Having a snapshot interval too large can create a situation where the database will hold the state it had back when the snapshot was taken.

A snapshot interval of a day is recommended and a week or more of snapshots can be useful if a problem remains on the system for a number of days and you wish to go back to a period prior to the point at which the problem existed.

`stored` monitors the database and is intelligent enough to refuse the latest snapshot if it suspects the database is not perfect. It will instead retrieve the latest most reliable snapshot. Despite the fact that a snapshot may be retrieved from a day ago, the system will use more up to date redundant data and override the older snapshot data, if available.

Thus, the ultimate role the snapshot plays is to get the system as close to up-to-date and ease the burden of the rest of the system trying to rebuild the data on the fly.

Message Store Database Snapshot Parameters

Parameter	Description
`local.store.snapshotpath`	Location of message store database snapshot files. Either existing absolute path or path relative to the `store` directory. Default: `dbdata/snapshots`
`local.store.snapshotdirs`	Number of different snapshots kept. Valid values: 2 -367 Default: 3

Message Store Command-line Utilities

Message Store Command-line Utilities

Topics:

List of Message Store Utilities

To manage the message store, Messaging Server provides a set of command-line utilities described in the following table.

Message Store Command-line Utilities

Utility	Description
`configutil`	Sets and modifies legacy configuration parameters.
`counterutil`	Displays all counters in a counter object. Monitors a counter object.
`deliver`	Delivers mail directly to the message store accessible by IMAP or POP mail clients.
`hashdir`	Identifies the directory that contains the message store for a particular user.
`imarchive`	Performs archiving actions on messages to support the AXS-ONE archiving system for Messaging Server.
`imcheck`	Prints mailbox data and metadata; also prints mboxlist database data for specified files and prints database statistics.
`imdbverify`	Takes and verifies message store database snapshots.
`imexpire`	Automatically removes messages from the message store based on administrator-specified criteria like age.
`iminitquota`	Reinitializes the quota limit from the LDAP directory and recalculates the disk space being used.
`immonitor-access`	Monitors the status of the Messaging Server components.
`impurge`	Purges unused cache records and message files in mailboxes.
`imquotacheck`	Calculates the total mailbox size for each user in the message store and compares the size with their assigned quota. Localized versions of `imquotacheck` notification incorrectly convert the % and the $ signs. To correct the encoding, replace every $ with \24 and replace every % with \25 in the message file.
`imsasm`	Handles the saving and recovering of user mailboxes.
`imsbackup`	Backs up stored messages.
`imsconnutil`	Monitors user access of the message store.
`imsexport`	Exports Messaging Server mailboxes into UNIX `/var/mail` format folders.

`imsimport`	Migrates UNIX /var/mail format folders into a Messaging Server message store.
`imsrestore`	Restores messages that have been backed up.
`imscripter`	The IMAP server protocol scripting tool. Executes a command or sequence of commands.
`mboxutil`	Lists, creates, deletes, renames, or moves mailboxes; reports quota usage.
`mkbackupdir`	Creates and synchronizes the backup directory with the information in the message store.
`moveuser`	Moves a user's account from one messaging server to another.
`msconfig`	Sets and modifies Unified Configuration options. Introduced in **Messaging Server 7 Update 5**.
`msgcert`	Generates a certificate request, adds a certificate to the certificate database, lists certificates in the database and so on. Starting with **Messaging Server 7 Update 4**, this command is considered deprecated. There are no plans to enhance this command and it might be removed in a future release. The `msgcert` command's key generation and certificate request capabilities are obsolete due to recent weakness in MD5 and the NIST 2010 guidelines for SSL security strength. Use `certutil` with appropriate options (`-Z SHA1 -g 2048`), or other third-party certificate generation tools, to create certificates and certificate requests with up-to-date security strength.
`msuserpurge`	Purges those user and domain mailboxes from the message store.
`readership`	Collects readership information on shared IMAP folders.
`reconstruct`	Reconstructs mailboxes that have been damaged or corrupted.
`refresh`	Refreshes the configuration of the specified messaging server processes
`rehostuser`	Enables you to move a Messaging Server user's mail store from one mailhost to another.
`relinker`	Consolidates duplicate messages.
`start-msg`	Starts the messaging server processes.
`stop-msg`	Stops the messaging server processes.
`stored`	Performs background and daily tasks, expunges, and erases messages stored on disk. Performs checkpoint database transactions, deadlock detection and rollback of deadlocked database transactions, cleanup of temporary files and lock files on startup, creation of a database snapshot archive, database recovery as necessary

Exit Codes

The message store utilities return exit codes from `sysexits.h`.

configutil

configutil

The `configutil` utility enables you to list and change Messaging Server configuration parameters.

For a list of all configuration parameters, see http://msg.wikidoc.info/index.php/Configutil_Reference.

All Messaging Server configuration parameters and values are stored locally in the `msg.conf` and `msg.conf.defaults` files. The `msg.conf.defaults` file must never be edited and contains defaults constructed during initial configuration. The `msg.conf` file contains settings that have been explicitly overridden from their default value using `configutil`. Use the `-H` option to `configutil` to view a setting's default value.Use `configutil` to edit `configutil` settings; do not edit these files directly.

> **Note**
> If the administrator has defined any language-specific options (such as messages), you must use the `language` option at the end of the command in order to list or change them. Commands entered without a `language` option are only applied to attributes that do not have a specified language parameter.

Requirements: Must be run locally on the Messaging Server. You may run `configutil` as `root` or `mailsrv`. If you make changes to the servers, you must restart or refresh the servers, depending on the variable, for the changes to take effect.

Location: *msg-svr-base*`/sbin/configutil`

You can use `configutil` to perform four tasks:

- Display particular configuration parameters using `-o` *option*.
  Add `;lang-`*xx* after the option to list parameters with a specified language parameter. For example, `;lang-jp` to list options specified for the Japanese language.

- List configuration parameter values using the `-p` *pattern* option. (Can be used with the `-m` option.)
  Use `-p` *pattern* to just list those configuration parameters whose names contain the pattern specified in *pattern*. `*` is the wildcard character and is assumed to follow *pattern* if there is no wildcard already in *pattern*.
  Use `-m` to show whether or not the listed parameters are refreshable.

- Set configuration parameters using the `-o` *option* and `-v` *value* options.
  Add `;lang-`*xx* after the option to set options for a specified language parameter. For example, `;lang-jp` to set options specified for the Japanese language.

- Import configuration parameter values from `stdin` using the `-i` option.
  Include the -H option to show settings with help.

  Examples:

```
# ./configutil -H

Configuration option: alarm.diskavail.msgalarmdescription
        Description: Description for the diskavail alarm.
        Syntax: string
        Default: percentage mail partition diskspace available
```

```
alarm.diskavail.msgalarmdescription is currently set to: percentage
mail
partition diskspace available

Configuration option: alarm.diskavail.msgalarmstatinterval
        Description: Interval in seconds between disk availability
checks. Set to 0 to disable checks of disk usage.
        Syntax: int
        Default: 3600
alarm.diskavail.msgalarmstatinterval is currently set to: 3600
[.....]

### Show help on all configuration parameters ending in ".port"

# ./configutil -p \*.port -H
Configuration option: local.ens.port
        Description: Port (and optionally, a specific IP address) ENS
server will listen on, in the format of [address:]port, for example,
7997 or 192.168.1.1:7997.
        Syntax: string
        Default: 7997
local.ens.port is currently set to: 7997

Configuration option: local.service.http.proxy.port
        Description: Configures the port number of the back-end
Messenger Express (HTTP) server with the Messaging Multiplexor.
        Syntax: uint
        Default: 80
local.service.http.proxy.port is currently set to: 80

Configuration option: local.snmp.port
        Description: SNMP subagent port number.
        Syntax: uint
        Default: 0
local.snmp.port is currently set to: 0

Configuration option: local.watcher.port
        Description: watcher listen port.
        Syntax: uint
        Default: 49994
local.watcher.port is currently set to: 49994

Configuration option: local.webmail.cert.port
        Description: Specifies a port number on the machine where the
Messaging Server runs to use for CRL communication. This port is used
locally for that machine only. The value must be greater than 1024.
        Syntax: int
        Default: 55443
local.webmail.cert.port is currently set to: 55443

Configuration option: local.webmail.da.port
        Description: Delegated Administrator port.
        Syntax: int
        Default: 8080
local.webmail.da.port is currently set to: 8080

Configuration option: local.webmail.sieve.port
        Description: The port of the web container where the Mail
Filter
```

```
has been deployed.
        Syntax: string
        Default: NULL (unset)
local.webmail.sieve.port is currently unset

Configuration option: metermaid.config.port
        Description: Port number on which MeterMaid listens for
connections.
        Syntax: tcpport
        Default: 63837
metermaid.config.port is currently set to: 63837

Configuration option: service.http.port
        Description: Messenger Express HTTP port.
        Flags: MSG_RESTART_HTTP
        Syntax: uint
        Default: 80
service.http.port is currently set to: 81

Configuration option: service.imap.port
        Description: IMAP server port number.
        Flags: MSG_RESTART_IMAP
        Syntax: uint
        Default: 143
service.imap.port is currently set to: 143

Configuration option: service.pop.port
        Description: POP server port number.
        Flags: MSG_RESTART_POP
        Syntax: uint
        Default: 110
service.pop.port is currently set to: 110

### Show help on store.partition.\*.path

# ./configutil -p store.partition.\*.path -H
Configuration option: store.partition.*.path
        Description: Controls the store index file directory path.
        Flags: MSG_RESTART_ALL
        Syntax: filepath
        Default: NULL (unset)
store.partition.primary.path is currently set
to: /opt/SUNWmsgsr/data/store/partition/primary
store.partition.three.path is currently set
to: /opt/SUNWmsgsr/data/store/partition/three
store.partition.two.path is currently set
to: /opt/SUNWmsgsr/data/store/partition/two

Configuration option: store.partition.primary.path
        Description: Full path name of the primary partition.
        Flags: MSG_RESTART_ALL
        Syntax: filepath
        Default: <msg.RootPath>/data/store/partition/primary
```

```
store.partition.primary.path is currently set
to: /opt/SUNWmsgsr/data/store/partition/primary
```

Syntax

```
configutil [-h] [-f <configfile>] [-o <option>[;<language>] [-v <value>]
configutil [-f <configfile>] [-p <pattern>] [-H] [-m] [-V]
configutil -i <inputfile>
```

Options

The options for this command are:

Option	Description
`-d`	Enables you to delete an option. Used with the `-o` option.
`-f` *configfile*	Enables you to specify a local configuration file other than the default. (This option uses information stored in the `CONFIGROOT` environment variable by default.)
`-h`	Shows usage statement.
-H	Enables you to get help on options. Used with the `-o` option.
`-i` *inputfile*	Imports configurations from a file. Data in the file to be entered in *option=value* format with no spaces on either side of the pipe. The *inputfile* should be specified as an absolute path.
`-m`	Lists meta data.
`-o` *option*	Specifies the name of the configuration parameter that you wish to view or modify. May be used with the `-H`, `-v`, `-d` options. Configuration parameter names starting with the word `local` are stored in the local server configuration file.
`-p` *pattern*	Lists only options with the given pattern (* is wildcard). Example: configutil {{- p *enable -H}}If no wildcard is present in the *pattern*, a wildcard at the end of the *pattern* will be assumed (effectively changing *pattern* to *pattern**).
`-v` *value*	Specifies a value for a configuration parameter. To be used with `-o` *option*.
`-V`	Enables you to validate the configuration against meta data.

If you specify no command-line options, all configuration parameters are listed.

Examples

To list all configuration parameters and their values in both the Directory Server LDAP database and local server configuration file:

```
configutil
```

To import configurations from an input file named `config.cfg`:

```
configutil -i config.cfg
```

To list all configuration parameters with the prefix `service.imap`:

```
configutil -p service.imap
```

To display the value of the `service.smtp.port` configuration parameter:

```
configutil -o service.smtp.port
```

To set the value of the `service.smtp.port` configuration parameter to 25:

```
configutil -o service.smtp.port -v 25
```

To clear the value for the `service.imap.banner` configuration parameter:

```
configutil -o service.imap.banner -v ""
```

To display the refreshable status of the `service.pop` configuration parameters:

```
configutil -m -p service.pop
```

This example of the -m option could produce the following sample output:

```
service.pop.allowanonymouslogin = no [REFRESHABLE]
service.pop.banner = "%h %p service (%P %V)" [REFRESHABLE]
service.pop.createtimestamp = 20030315011827Z [REFRESHABLE]
service.pop.creatorsname = "cn=directory manager" [REFRESHABLE]
service.pop.enable = yes [NOT REFRESHABLE]
service.pop.enablesslport = no [NOT REFRESHABLE]
service.pop.idletimeout = 10 [REFRESHABLE]
service.pop.maxsessions = 600 [NOT REFRESHABLE]
service.pop.maxthreads = 250 [NOT REFRESHABLE]
```

Language Specific Options

To list or set options for a specific language, append `;lang-`*xx* immediately after the option with no spaces, where *xx* is the two-letter language identifier. For example, to view the text of the Japanese version of the `store.quotaexceededmsg` message:

```
configutil -o "store.quotaexceededmsg;lang-jp"
```

The semicolon is a special character for most UNIX shells and requires special quoting as shown in the example.

counterutil

counterutil

The `counterutil` utility displays and changes counters in a counter object. It can also be used to monitor a counter object every 5 seconds. For more information and examples see Gathering Message Store Counter Statistics Using counterutil

Requirements: Must be run locally on the Messaging Server.

Location: *msg-svr-base*/`sbin`/

Syntax

```
counterutil -l

counterutil -o _counterobject_ [-i _interval_] [-n _numiterations_]
counterutil -s -o _counterobject_ -c _counter_
```

Options

The options for this command are:

Option	Description
`-c` *counter*	Specifies a particular counter associated with a counter object.
`-i` *interval*	Specifies, in seconds, the interval between reports. The default is `5`.
`-l`	Lists the content of the counter registry.
`-n` *numiterations*	Specifies the number of iterations. The default is infinity.
`-o` *counterobject*	Displays the contents of a particular counter object at specified intervals (by default, every 5 seconds). The valid counter objects are:`imapstat`, `popstat`, `httpstat`, `alarm`, `diskusage`, `serverresponse`. Each counter object has a list of counters. When you monitor a counter object, `counterutil` displays the counters associated with it.
`-s`	Resets the counter to 0.

Examples

To list all counter objects in a given server's counter registry:

```
counterutil -l
```

To monitor the content of a counter object `imapstat` every 5 seconds:

```
counterutil -o imapstat
```

To reset the counter `global.maxconnections`, associated with the counter object `imapstat`, to zero:

```
counterutil -s -o imapstat -c global.maxconnections
```

For usage information on `counterutil`, refer to the To Gather Message Store Counter Statistics Using counterutil

hashdir

hashdir

The `hashdir` command calculates the hash where the specified user ID would be found in a store partition. If the user is not in the default domain, you should append @<domain> to the user ID.

Syntax

```
hashdir [-a] [-i] <userid>[@<domain>]
```

Options

The options for this command are:

Option	Description
`-a`	Appends the specified user ID to the output.
`-i`	Allows you to use the command in interactive mode.

Examples

```
# hashdir user1
64/b1/
# hashdir user1@domain.com
11/05/
```

Note
The `hashdir` command does not validate the input to determine whether it is a valid userid or whether the mailbox exists in the store. Also, the user ID is case sensitive, specifying the wrong case will generate a different hash. To see where the folder exists in the store, the mboxutil -lxp user/<userid> command may be more useful.

imarchive

imarchive

The `imarchive` utility supports the AXS-One archiving system for Messaging Server. `imarchive` reads the AXS-One archive report files and performs one of the following actions:

- Marks the messages as archived in the message store. Messages remain in both the message store and archive system. Marking a message prevents it from being re-archived. Marking messages is the default action.
- Creates URL stubs of the messages. The user will see the same header, internal date, save date, and message flags, but instead of the message text, the user will have a clickable URL to retrieve the message text.

Requirements: Must be run locally on the Messaging Server. You can run `imarchive` as `root` or as `mailsrv`.

Location: *msg-svr-base*`/sbin/`

Syntax

```
imarchive [-s] [-v] [-p <dir>]
```

Options

The options for this command are:

Option	Description
`-p` *dir*	Processes archive reports in the specified directory
`-s`	Replaces archived messages with stubs
`-v`	Provides verbose logging information

If you do not use the `-s` option, `imarchive` takes the default action, marking the messages as archived in the message store.

Example

To stub messages:

```
imarchive -s
```

imcheck

imcheck

The `imcheck` utility prints mailbox data and metadata. In addition, `imcheck` prints mboxlist database data for specified database files and prints database statistics.

Syntax

```
imcheck [-m mailbox [-c|-b messagenum] | -x dir [-c|-b messagenum] | -p
partition |
-f file] [-e | -H] [-D [-S separator]]

imcheck -q

imcheck -d <databasename> [-S separator]

imcheck -s [-n] [subsystem...]
```

Options

The options for this command are:

Option	Description
`-b` *messagenum* This option is available starting with **Messaging Server 7.0.5.28.0**.	Prints the contents of the message specified by *messagenum*.
`-c` *messagenum*	Prints cache data for the specified message number.
`-d` *databasename*	Prints the contents of the database specified by *databasename*.
`-D`	Prints metadata in `imsbackup` format to verify the success of the backup operation.
`-e`	Prints the uids of expunged messages.
`-f` *file*	Prints metadata for the mailboxes listed in the specified file. You must specify a full path name for *file*.
`-m` *mailbox*	Prints metadata for the specified *mailbox*. The *mailbox* name must be in Modified UTF-7 encoded format. You can use the `"-E M-UTF-7"` option with the `mboxutil` command to list mailboxes in Modified UTF-7 format. For information about the output produced by this option, see imcheck -m Metadata Output

`-n`	Specifies that no locking will occur. This option should be used only for debugging database hang problems. **Caution** Only use this option when the database is hung. It can cause data corruption in the message store.
`-p` *partition*	Prints metadata for every mailbox in the specified message store partition.
`-q`	Displays the maintenance queue. See Displaying the Maintenance Queue for more information.
`-s`	Prints database statistics.
`-S` *separator*	Specifies an output field separator. The default value is a vertical bar: "|"
`-x` *dir* This option is available starting with **Messaging Server 7.0.5.28.0**.	Prints metadata for the mailbox under the directory specified by *dir*.
`subsystem`	Prints the replication statistics. `subsystem` can be `spool`, `lock`, `log`, `txn`, or `rep`.

`imcheck -m` Metadata Output

The `imcheck -m` *mailbox* command displays metadata that describes the entire mailbox and metadata that describes each message.

The message-specific data is displayed in a table. Most of the fields in the table are self-explanatory. However, the following fields need further explanation:

HSize - Header size

MT - Message type ID, defined by `configutil` parameters such as `store.messagetype.*` and `store.messagetype.enable`. For a list of these parameters, see http://msg.wikidoc.info/index.php/Configutil_Reference.

SFlags - System flags. The system flags are as follows:

```
R : Recent
S : Seen
D : Deleted
A : Answered
F : Flagged
T : Draft
B : Stubbed
C : Archived
E : Encrypted
N : NoLeadingDots
```

In the `imcheck -m` *mailbox* output, the system-flag metadata is displayed as a character. For example:
S
indicates that the Seen flag has been set.

UFlags - Flags defined by the user. User-flags are displayed as a hex number. The binary form of the hex number represents the user-flag switches. For example, if the user has defined six flags, the value
`3f0000`
indicates that all six flags are set.

Examples

To dump metadata to verify the `imsbackup` operation performed on the user `jdoe`'s INBOX, separating the output with a colon (":"):

```
imcheck -D -S ":" -m user/jdoe/INBOX
```

To dump the contents of the `folder.db` database:

```
imcheck -d folder.db
```

To print metadata for the inbox of the user `jdoe`:

```
imcheck -m user/jdoe/INBOX
-------------------------------------------------------------
Name: user/jdoe
Version: 102
Exists: 10
Flags: 0
Largest Msg: 1094 bytes
Last Append: 20080801062616
Last UID: 1008276527
Oldest Msg: 20000621093214
Oldest Uid: 2
Quota Used: 10061
UID Validity: 1008099930
Cache Offset: 7856
Start Offset: 256
ACL: jdoe lrswipcda
Userflags:
f1
f2
f3
f4
f5
f6
Subscribed: 0
Partition: primary
Path: /var/opt/SUNWmsgsr/store/partition/primary/=user/64/b1/=jdoe
Msg Path: /var/opt/SUNWmsgsr/store/partition/primary/=user/64/b1/=jdoe

MsgNo Uid Internal-Date Sent-Date Size HSize Cache-Id C-Offset C-Len
Last-Updated Save-Date MT SFlags UFlags Original-Uid Message-id
------------------------------------------------------------------------
1 20080801061303 20080801061259 752 744 1 16 1324 20080801061303
20080801061303 0 S 0.0.0 1217596383-1
<200808011312.m71DCxJp017783@dumbo.sesta.com>
2 2 20080801062616 20080801062615 781 772 1 1340 1440 20080801062616
20080801062616 0 S 0.0.0 1217596383-2
<200808011326.m71DQFYb018990@dumbo.sesta.com>
```

To print metadata for the Sent folder of a Japanese User `jauser`:

```
bash-3.00# ./mboxutil -E "M-UTF-7" -lp user/jauser/*
msgs Kbytes last msg partition quotaroot mailbox

0 0 - - primary 5120 user/jauser/INBOX
0 0 - - primary - user/jauser/&MFQwf3ux-
0 0 - - primary - user/jauser/&Tgtm+DBN-
1 0 2009/03/23 11:59 primary - user/jauser/&kAFP4W4IMH8-
bash-3.00# ./imcheck -m "user/jauser/&kAFP4W4IMH8-"
--------------------------------------------------------------------------------
user/jauser/&kAFP4W4IMH8-
Version: 102
Exists: 1
Flags: 0
Largest Msg: 898 bytes
Last Append: 20090323115905
Last UID: 1
Oldest Msg: 20090323115905
Oldest Uid: 1
Quota Used: 898
UID Validity: 1237768457
Cache Offset: 7856
Start Offset: 256
ACL: jauser lrswipcda
Subscribed: 1
Partition: primary
Path:
/opt/SUNWmsgsr/data/store/partition/primary/=user/20/b0/=jauser/=&k+A+F+P4+
Path:
/opt/SUNWmsgsr/data/store/partition/primary/=user/20/b0/=jauser/=&k+A+F+P4+
Uid Internal Date Sent Date Size HSize CacheOff Last Updated Save Date
MT SFlags UFlags Original-Uid Message-id
--------------------------------------------------------------------------------
1 20090323115905 20090323115905 898 549 7856 20090323115905
20090323115905 0 S 0.0.0 1237768457-1
<cd8b67384048.49c77989@dumbo.sesta.com>
```

imexpire

imexpire

`imexpire` automatically expires messages in the message store based on administrator-specified criteria. With Messaging Server 5.x and 6.x, `imexpire` was executed in two phases. The first phase expires messages that meet the expire criteria; the second phase purges (removes) from the message store all messages that have been discarded by `imexpire` or expunged by users. With Messaging Server 7, the second phase is performed by the impurge daemon process.

Topics:

Expire Actions

The expire phase can perform one of these actions:

- Discard – removes messages from the mailbox immediately.
- Archive – archives messages by using the AxsOne archive store.
- Fileinto – moves messages to a specified mailbox folder.
- Report – prints the specified mailbox name, `uid`, and `uid` validity to `stdout`.

By default, `imexpire` discards messages.

For details about how the `imexpire` actions can be performed, see Message Store Maintenance Queue , Configuring Message Expiration, and Message Store Message Expiration.

Expire Criteria

The expire criteria can be set with `configutil` parameters or in a file called `store.expirerule`. Here are some of the criteria that can be specified:

- Folder pattern
- Number of messages in the mailbox
- Total size of the mailbox
- Age, in days, that messages have been in the mailbox. This criterion dates the age of a message from the time it first arrives in the message store (is first received by the user).
- Age, in days, that messages have been saved in a particular mailbox folder. This criterion dates the age of a message from the time it is moved to a particular folder or re-saved in a folder.
- Size of message and grace period (days that a message exceeding the size limit will remain in the message store before removal)
- Whether a message has been flagged as **seen** or **deleted**
- By message header field such as subject or message ID
- According to a sieve script, as defined in RFC 3028

You can use `imexpire` to install a local expire rule file (`store.expirerule`) without conflicting with existing expire rules. If an expire rule file configured for the same partition or mailbox is executing while you try to install a new expire rule file, a warning message appears and the new expire rule file is not installed. Use the `imexpire -i` option to install a local expire rule file.

You can exclude a particular user or mailbox folder from all expire criteria by setting the `exclusive` expire rule for that user or mailbox without specifying any other rules in the expire rule file.

> **Note**
> The functionality of `imexpire` has been expanded and the interface has changed since earlier versions of Messaging Server. However, this version continues to support older `imexpire` configurations.

Requirements

`imexpire` must run locally on the Messaging Server; the `stored` utility must also be running.

Location

The location of `imexpire` is *msg-svr-base*`/sbin`.

Syntax

```
imexpire  [-n] [-d] [-v <num>] [-p <partition> | -u <user>] [-t <num>][-r
<num>] [-m <num>] [-f <file>]

imexpire -i {-p <partition> | -x <mailbox>} -f <file>
```

Options

The options for this command are:

Option	Description
-f *file*	Use the expire rules specified in *file*. All other expire rules are ignored. When used with the `-i` option, `-f` *file* refers to the expire-rule file to be installed. Use a full path name to specify *file*. The expire rules in *file* must use the same format as the rules in the global expire configuration file.
-i	Install a local expire-rule configuration file. This option must be used with either the `-p` *partition* option to specify a message store partition or the `-x` *mailbox* option to specify a mailbox. In addition, it must be used with the `-f` *file* option to specify the expire rule file to install.
`-n`	Trial run only – do not perform expire. A description of what would happen without this flag is output.
`-v 0|1|2`	Log expire statistics. The number specifies the log level, where 0 = store level statistics (default setting) 2 = mailbox level statistics 3 = message level statistics Messages are logged to the log file by default. When the `-d` option is used, messages go to `stdout`.
`-d`	Display debug output to `stdout/stderr`.
`-p` *message_store_partition*	Expire the specified message store partition.
`-u` *user*	Expire the specified user.
`-t` *num*	Maximum number of threads per process. Default is `50`.
`-r` *num*	Maximum number of threads per partition. Default is `1`.
`-m` *num*	Maximum number of rules in a policy. Default is `128`.
`-x` *mailbox*	Name of the mailbox to which the local expire rules apply. For example: `user/joe/INBOX`. This option is used with the `-i` option to install a local configuration file that will expire messages in the specified mailbox and its sub-folders.

Examples

Install a local expire rule configuration file for the user jdoe. These expire rules will apply to jdoe's memos folder.

```
imexpire -i -x user/jdoe/memos -f /home/jdoe/store.expirerule
```

Enhancements Introduced in Messaging Server 7 Update 5

Messaging Server 7 update 5 provides the following enhancements to `imexpire`:

- New Attributes for Spam and Virus Scanning Through an MTA Channel
- New Sieve Body-Test Functionality

New Attributes for Spam and Virus Scanning Through an MTA Channel

To facilitate spam and virus scanning through an MTA channel, the following attributes have been added to `imexpire`:

Attribute	Description
`channel`	An MTA channel.
`rescanhours`	Rescan messages that have not been scanned for the specified number of hours.

New Sieve Body-Test Functionality

Sieve body-test functionality has been added to the `imexpire` utility. The test can find and perform actions on existing email messages in the Message Store on keywords in the body part.

To enable a Sieve body-test, set `ENABLE_SIEVE_BODY=1` in `option.dat`.

Example usage in `store.expirerule`:

folderpattern: *
sieve: require "body"; body :contains "bug";
action: discard

iminitquota

iminitquota

The `iminitquota` utility reinitializes the quota limit from the LDAP directory and recalculates the total amount of disk space that is being used by the users. It updates the message store `quota.db` database under the `mboxlist` directory in the message store. The `iminitquota` utility should be run after the `reconstruct -q` utility is run.

Location: *msg-svr-base*/`sbin`/

Syntax

```
iminitquota -a | -q | -s | -u <userid>
```

Options

The options for this command are:

Option	Description
`-a`	Initializes and updates the quota files for every message store user.
`-q`	Initializes quota only.
`-s`	Initializes quota, usage, and overquota status.
`-u` *userid*	Reinitializes and updates the quota-related information for the specified user. The *userid* parameter specifies the message store id of a user, not the login id of the user.

You must specify either the `-a` or `-u` option with the `iminitquota` command.

immonitor-access

immonitor-access

Monitors the status of Messaging Server components—Mail Delivery (SMTP server), Message Access and Store (POP and IMAP servers), Directory Service (LDAP server) and HTTP server. This utility measures the response times of the various services and the total round trip time taken to send and retrieve a message. The Directory Service is monitored by looking up a specified user in the directory and measuring the response time. Mail Delivery is monitored by sending a message (SMTP) and the Message Access and Store is monitored by retrieving it. Monitoring the HTTP server is limited to finding out weather or not it is up and running.

The internal operation of `immonitor-access` is as follows: first it does an ldapsearch of a test user created by the administrator. This checks the Directory Server. It can then connect to the SMTP port and send a message to the mail address to check the dispatcher. Then, it checks Message Access by using the IMAP and POP server to see if the message made it to the Message Store. The command logs a message in the default log file if any of the thresholds are exceeded.

The command creates a report that contains the following information:

- The state of the components
- The response time
- The round-trip time for that service

`immonitor-access` is typically run by `cron` at scheduled intervals to provide a snapshot of the status of the Message Access and Store components. `immonitor-access` can also connect to the IMAP/POP service and delete messages with the subject specified by `-k`. If `-k` is not specified, all messages containing the subject header, `immonitor`, are deleted.

The administrator must create a test user for use by this command before it can be executed.

Syntax

```
immonitor-access -C <LMTP_host>[:<port>]=[<threshold>][,STLS|PORT] -u
<user_name>
[-D <threshold>] [-m <test_file>] [-k <subject>] [-dhnv]
[-A <alert_host>] [-f <alert_from>] [-r <alert_recipients>] [-X] [-T]

immonitor-access -I <IMAP_host>[:<port>]=[<threshold>][,STLS|PORT]
-u <user_name> [-w passwd | -j pwdfile]
[-D <threshold>] [-k <subject>] [-dhnvz]
[-A <alert_host>] [-f <alert_from>] [-r <alert_recipients>] [-X] [-T]

immonitor-access -H <HTTP_host>[:<port>]=[<threshold>][,STLS|PORT] [-c
<cookiename>]
-u <user_name> [-w passwd | -j pwdfile]
[-D <threshold>] [-dhnv]
[-A <alert_host>] [-f <alert_from>] [-r <alert_recipients>] [-X] [-T]

immonitor-access -L <LDAP_host>[:<port>]=[<threshold>][,STLS|PORT] -u
<user_name>
[-b searchbase -B bindDN] [-w passwd | -j pwdfile]
[-D <threshold>] [-r <alert_recipients>] [-A <altenate_host>]
[-f <alert_from>] [-X] [-T] [-dhnv]

immonitor-access -P <POP_host>[:<port>]=[<threshold>][,STLS|PORT]
-u <user_name> [-w passwd | -j pwdfile]
[-D <threshold>] [-k <subject>] [-dhnvz]
[-A <alert_host>] [-f <alert_from>] [-r <alert_recipients>] [-X] [-T]

immonitor-access -S <SMTP_host>[:<port>]=[<threshold>][,STLS|PORT] -u
<user_name>
[-D <threshold>] [-m <test_file>] [-k <subject>] [-dhnv]
[-A <alert_host>] [-f <alert_from>] [-r <alert_recipients>] [-X] [-T]
```

Options

The following list contains valid task options for the command.

Option	Description
`-u` *user_name*	The valid test user account to use. This test mail user has to be created by the administrator. If the test mail user is in a hosted domain, *user@domain* should be specified.
`-w` *passwd*	The password corresponding to the user specified with `-u`. To read the input from standard input, `--` can specified with `-w`. ANY PASSWORD SPECIFIED ON THE COMMAND LINE WITH `-w` WILL BE VISIBLE TO OTHER USERS OF THE SYSTEM. Use of the `-j` option is strongly encouraged. This option, or `-j`, is mandatory when the `-H`, `-I`, or `-P` options are used, or when `-n` is used in conjunction with `-L`.
`-j` *pwdfile*	A readable file containing the password corresponding to the user specified with `-u`. This option, or `-w`, is mandatory when the `-H`, `-I`, or `-P` options are used, or when `-n` is used in conjunction with `-L`. This might be available in a future release.

`-C` *LMTP_host* `:` [*port*] = [*threshold*]	Use the LMTP server and the port specified to check if Messaging Server is able to deliver the message to the store. The threshold is specified in seconds.
`-H` *HTTP_host* `:` [*port*] = [*threshold*]	Use the HTTP server and the port specified to check if the HTTP server is able to accept requests on the specified port. When `-I -H` or `-P` is used, it is necessary to provide the test user password with `-w`. When `-S`/`-C`, `-I`/`-P` are specified together, the command does the following: + sends mail and retrieves with IMAP and POP + reports the per protocol response time + reports round-trip time o reports delivery time (the time taken to send the mail and be visible to IMAP/POP) Multiple `-I`, `-P`, and `-S` options can be specified, which helps in monitoring Messaging Server on various systems.
`-I` *IMAP_host* `:` [*port*] = [*threshold*]	Use the IMAP server and the port specified to check the IMAP component of the Message Access. The threshold is specified in seconds. The threshold involves the time to login, retrieve, and delete the message.
`-L` *LDAP_host* `:` [*port*] = [*threshold*]	Use the LDAP server and the port specified to check the Directory Server. The threshold is specified in seconds.
`-P` *POP_host*`:` [*port*] = [*threshold*]	Use the POP server and the port specified to check the POP component of the Message Access. The threshold is specified in seconds. The threshold involves the time to login, retrieve, and delete the message.
`-S` *SMTP_host* `:` [*port*] = [*threshold*]	Use the SMTP server and the port specified to check if Messaging Server is able to accept mail for delivery. The threshold is specified in seconds.
`-A` *alert_host*	The SMTP server to send e-mail alerts to. This option helps in sending alert messages even when the primary mail server is down or heavily loaded. If `-A` is not specified, the SMTP server on the localhost is used.
`-b` *searchbase*	Use search base as the starting point for the searching in the Directory Server. It is the same as `-b` of `ldap-search(1)`. If `-b` is not specified, the utility uses the value of dcRoot of the configuration parameter `local.ugldapbasedn`.
`-B` *bindDN*	LDAP DN to bind as when performing an LDAP search via `-L`. If not specified, then the value of configuration parameter `local.ugldapbinddn` is used. This option as well as `-j` or `-w` is mandatory when the `-n` option is used.
`-d`	The debug mode: display the execution steps.
`-D` *threshold*	The delivery (also called round-trip time) threshold. The time taken to send the mail and the mail being visible to POP/IMAP. This option can be used only when `-I`/`-P` and `-S`/`-C` are used.
`-f` *alert_from*	When `immonitor-access` sends out an e-mail, it usually is sent as root@ *domainname*. Specify this option to send out an e-mail as different user: `-f user@red.iplanet.com`
`-h`	Prints command usage syntax.
`-k` *subject*	Header subject of the messages to be sent/deleted. The utility, by default, uses the string “immonitor:<date>” as the subject in the header sent out with the `-S` option. If `-k` is specified, the string “immonitor:subject” is used in the subject header. This option can be used with `-z` to delete messages, if `-k` is not specified, all messages with the Subject header containing “immonitor” are deleted.

`-m` *test_file*	The file that is mailed to the test user. You can get response and round-trip times for various mail sizes with this option. Specify only text files as non-text files result in unexpected behavior. If `-m` is not specified, the `mailfile.txt` file in *msg-svr-base* `/lib/locale/C/mailfile.txt` is used as the mail file.
`-n`	Operate without a Messaging Server configuration. Useful when running the utility on a remote system. This might be available in a future release.
`-r` *alert_recipients*	A comma-separated list of mail recipients who will be notified. If this option is not specified, the command reports the alert messages on the standard output.
`-v`	Run in verbose mode, with diagnostics written to standard output.
`-z`	Delete messages containing the string specified by `-k` in the subject header. If `-k` is not specified, all messages with the subject header containing “immonitor” are deleted. Use `-z` only with `-I` or `-P`. Do note use `-z` with `-S` or `-C` as this can cause unexpected results.
`-X`	Enable SASL.
`-T`	Enable SSL.

The default ports are:

SMTP = 25
IMAP = 143
POP = 110
LDAP = 389
LMTP = 225
HTTP = 80

If either the port or threshold is not specified, default ports with the default threshold of 60 seconds is assumed. The threshold specified can be a decimal number.

Output

The command generates a report containing the various protocol execution times. For example:

```
Smtp Statistics for: thestork:25
Connect Time: 2.122 ms
Greeting Time: 5.729 ms
Helo Time: 2.420 ms
Mail From: Time: 2.779 ms
Rcpt To: Time: 4.128 ms
Data Time: 1.268 ms
Sending File Time: 94.156 ms
Quit Time: 0.886 ms
Total SMTP Time: 113.488 Milliseconds
```

If the alert recipients are specified and any of the threshold values are exceeded, the command mails the report containing the service name and the response time:

```
ALERT: <service> exceeds threshold Response time=secs/Threshold=secs
```

Note that in case of times reported for IMAP, the individual times might not add up to the exact value shown by the “Total IMAP time”. This occurs because the message does not get to the store immediately. The utility loops until the message is found. Typically, the search time indicates only the

successful search time. However, the total time includes each of the individual sleep and search times.

With POP, the utility needs to login and logout multiple times before the message is actually found in the store. Thus, the total time here is the accumulated time for all the logins and log outs.

Example 1: To monitor the SMTP, IMAP, and POP with the threshold 250 milliseconds more than the default value (60 seconds) on localhost use:

```
immonitor-access -S localhost:=60.25 -I localhost:=60.25 -P localhost:=60.25
-u test_user -w passwd
```

This example assumes that `test_user` exists with password "`passwd`."

Example 2: To monitor LDAP on localhost with no Messaging Server configuration use:

```
immonitor-access -n -L 127.0.0.1:389 -b o=usergroup -u test_user -B
"cn=directory manager" -w passwd
```

The above example assumes that the Directory Manager password is `passwd` and that there is a user named `test_user`.

Example 3: To check remote connectivity and latency issues on `remotehost` use:

```
immonitor-access -S remotehost:=10 -I remotehost:=10 -P remotehost:=10 -u
tester -w passwd
```

Example 4: To run `immonitor-access` on a normal port use:

```
immonitor-access -u admin -w password -P host:110
```

Example 5: To run `immonitor-access` using STLS:

```
immonitor-access -u admin -w password -P host:110 -X
immonitor-access -u admin -w password -P host:110=STLS -X
```

Example 6: To run `immonitor-access` using SSL via STLS:

```
immonitor-access -u admin -w password -P host:110=STLS -T -X
```

Example 7: To run `immonitor-access` on the SSL port:

```
immonitor-access -u admin -w password -P host:995 -T
immonitor-access -u admin -w password -P host:995=PORT -T
```

Example 8: To run `immonitor-access` using SASL on SSL:

```
immonitor-access -u admin -w password -P host:995 -T -X
immonitor-access -u admin -w password -P host:995=PORT -T -X
```

Exit Status

The exit status is 0 if no errors occur. Errors result in a non-zero exit status and a diagnostic message being written to standard error. A different exit status is returned when various thresholds are exceeded.

0	Successful execution with no errors or thresholds exceeded
1	Exceeded threshold of a service
2	Errors
64	Usage errors
75	Insufficient virtual memory

An alert message is written to the console when the response time of any server exceeds the threshold.

An error message is written to the console when any of the servers cannot be reached.

Warnings

The password passed with `-w` can be visible to a user using the `ps(1)` command. It is strongly advised that you create a test user to be specifically used by the monitoring utilities.

It is recommended that you use `-w` and enter the password through standard input. However, if the utility is executed through `cron`, the password can be stored in a file. This file can be redirected as the standard input for the utility.

```
cat passwd_file | immonitor-access -w -
immonitor-access -w - ... < passwd_file
```

Do not use the echo command such as:

```
echo password | immonitor-access .. -w - ..
```

because the `ps` might show the echo's arguments.

To delete the test mail sent by the `-S` option, invoke the `immonitor-access` command with the `-z` option separately. Do not use the two together.

imquotacheck

imquotacheck

The `imquotacheck` utility administers user quotas and domain quotas in the message store. This utility can also compare mailbox size with a user's assigned quota. As an option, you can email a notification to users who have exceeded a set percentage of their assigned quota.

The `imquotacheck` utility lists users in the local mboxlist database. To list users in the LDAP directory, use the `imquotacheck -a` option.

Requirements: Must be run locally on the Messaging Server.

Location: *msg-svr-base*`/sbin/`

Syntax

To report quota and usage information to `stdout`:

```
imquotacheck [-u <user> | -d <domain> [-p <partition>] | -p <partition> ]
[-a]
```

To enforce a domain quota:

```
imquotacheck -f [-d <domain>]
```

To notify users when they have exceeded a set percentage of their assigned quota (to deliver over-quota notification messages):

```
imquotacheck -n [-e] [-d <domain>] [-r <rulefile>] [-t <message template>]
[-l]
```

Options

The options for this command are:

Option	Description
-a	Lists users in the LDAP directory. If the -a option is not used, imquotacheck lists users in the message store (the local mboxlist database).
-d <domain>	Searches for users only in the specified domain.When used with the -f option, the -d <domain> option enforces quotas for the specified domain.When used with the -n option, the -d <domain> option delivers over-quota notifications for users in the specified domain.
-f	Enforces domain quotas. If the domain is over quota and the mailDomainStatus attribute is currently set to active, the value will be reset to overquota, which will prevent mail from being accepted by the message store. If the domain is not over quota and the mailDomainStatus attribute is set to overquota, then the value will be changed to active, and mail will be accepted.
-h	Displays help for this command.
-n	Sends notification messages based on the rules defined in the <rulefile>. If you do not define any rules with the -r option when you use the -n option, the default <rulefile> is used. If you have not set up a default <rulefile> and you do not define any rules when you use the -n option, you do not receive an error. Use of imquotacheck -n is mutually exclusive with dynamic notification (that is, when store.quotanotification is enabled). See Methods of Notification.
-p <partition>	Reports quota for the specified partition.
-r <rulefile>	Specifies the set of rules to be used when you want to calculate quota usage. This option is used with the -n option. If -r is not specified, a default <rulefile> can be used. To set up a default <rulefile>, copy the Sample Rulefile to <msg-svr-base>/config/imq.rulefile. See Rulefile Format.
-t <message template>	Notifies users when their mailbox quota is exceeded. This option is used with the -n option.The message template format is the following:* %U% - user's mailbox id* %Q% - percentage of the used mailbox quota* %R% - quota usage details: assigned quota, total mailbox size, and percentage used. If the -e is specified, mailbox usage of the individual folders are also reported.* %M% - current mailbox size* %C% - quota attribute valueIf -t is not specified, a default message file will be mailed. To setup a default message file, copy the Notification File to <msg-svr-base>/config.
-u <user>	Obtains the quota status of the specified user id. Cannot be used to specify multiple users.
-e	report disk usage of the individual folders in notification messages
-l	log users to action log who are above quota

Examples

To send a notification to all users in accordance to the default rulefile:

```
imquotacheck -n
```

To send a notification to all users in accordance to a specified rulefile, myrulefile, and to a specified mail template file, mytemplate.file:

```
imquotacheck -n -r myrulefile -t mytemplate.file
```

To enforce the domain quota for the `siroe.com` domain:

```
imquotacheck -f -d siroe.com
```

To list the usage of all users whose quota exceeds the least threshold in the `rulefile`:

```
imquotacheck
```

To list per folder usages for user `user1`:

```
imquotacheck -u user1
```

To only list the users in domain `siroe.com`:

```
imquotacheck -d siroe.com
```

Rulefile Format

The `rulefile` format is organized into two sections: a general section and a rule name section. The general section contains attributes that are common across all rules. Attributes that are typically specified in the general section are the `mailQuotaAttribute` and the `reportMethod`. In the rule name section, you can write specific quota rules for notification intervals, trigger percentages, and so on. Attributes that are typically specified in the rule name section are `notificationTriggerPercentage`, `enabled`, `notificationInterval`, and `messageFile`. Note that the attributes and corresponding values are not case-sensitive. The following rulefile format is used:

```
[General]
mailQuotaAttribute = [value]
reportMethod = [value]

[rulename1]
attrname=[value]
attrname=[value]

[rulename2]
attrname=[value]
attrname=[value]

[rulename3]
attrname=[value]
attrname=[value]
```

The following table shows the attributes, whether they are required, the default value, and the description.

rulefile Attributes

General Attribute	Required Attribute?	Default Value	Description
`mailQuotaAttribute`	No	Value in `quotadb`	Specifies the name of the custom mailquota attribute. If not specified, the value in `quotadb` is used.
`reportMethod`	No		You can provide your own code to customize the output of the quota report. The value of this attribute is specified as <library-path:function>, where <library-path> is the path of the shared library and <function> is the name of the report function. See `reportMethod` Signature for more information about what parameters your function must accept and return.

Rule Attribute	Required Attribute?	Default Value	Description
`notificationTriggerPercentage`	Yes		Specifies the consumed quota percentage that will trigger notification. Value should be unique and an integer.
`messageFile`	No	<msg-svr-base> `/config/imq.msgfile`	Specifies the absolute path to the message file.
`notificationInterval`	Yes		Indicates the number of hours before a new notification is generated.
`enabled`	No	`0` (FALSE)	Indicates if the particular rule is active. Applicable values are `0` for FALSE and `1` for TRUE.
`notificationMethod`	No		You can provide your own code to perform the overquota notification. The value of this attribute is specified as <library-path:function>, where <library-path> is the path of the shared library and <function> is the name of the report function. See `notificationMethod` Signature for more information about what parameters your function must accept and return.

`reportMethod` Signature

If you override the imquotacheck `reportMethod()` with your own function, it must be defined as:

```
int symbol(QuotaInfo* info, char** message, int* freeflag)

typedef struct QuotaInfo {
const char* username; /* user name (uid or uid@domain) */
long quotakb; /* quota in kbytes */
long quotamsg; /* quota in number of messages */
ulong usagekb; /* total usage in kbytes */
ulong usagemsg; /* total usage in number of messages */
FolderUsage* folderlist; /* folder list (for -e) */
long num_folder; /* number of folders in the folderlist */
long trigger; /* not used */
const char* rule; /* not used */
}

typedef struct FolderUsage {
const char*foldername;
ulong usagekb; /* folder usage in kbytes */
}
```

The address, `message`, points to the output message. The report function is expected to fill the value of `*message` and allocate memory for `message` when necessary. The `freeflag` variable indicates if the caller is responsible for freeing allocated memory for `*message`.

The return values are 0 for success and 1 for failure.

The `imquotacheck` function will invoke the `reportMethod` to generate the report output. If the `reportMethod` returns 0 and `*message` is pointing to a valid memory address, `message` will be printed.

If the `*freeflag` is set to 1, the caller will free the memory address pointed to by `message`. If the `-e` option is specified, the quota usage for every folder will be stored in the `folderlist`, an array in `FolderUsage`; the `num_folder` variable is set to the number of folders in the `folderlist`.

`notificationMethod` Signature

If you override the imquotacheck `notificationMethod()` with your own function, it must be defined as:

```
int symbol(QuotaInfo* info, char** message, int* freeflag)

typedef struct QuotaInfo {
const char* username; /* user name (uid or uid@domain) */
long quotakb; /* quota in kbytes */
long quotamsg; /* quota in number of messages */
ulong usagekb; /* total usage in kbytes */
ulong usagemsg; /* total usage in number of messages */
FolderUsage* folderlist; /* folder list (for -e) */
long num_folder; /* number of folders in the folderlist */
long trigger; /* the exceeded notificationTriggerPercentage */
const char* rule; /* rulename that triggered notification */
}

typedef struct FolderUsage {
const char *foldername;
ulong usagekb; /* folder usage in kbytes */
}
```

The address, `message`, points to the notification message. The notification function is expected to fill in the value of this variable and allocate the memory for the message when necessary. The `freeflag` variable indicates if the caller is responsible for freeing the memory allocated for `message`.

The return values are 0 for success and 1 for failure.

If the notification function returns a 0, and `*message` is pointing to a valid address, the `imquotacheck` utility will deliver the message to the user. If the `*freeflag` is set to 1, the caller will free the memory address pointed to by message after the message is sent.

If the `-e` option is specified, the quota usage for every folder will be stored in the `folderlist` variable, an array of `FolderUsage` structure; the `num_folder` variable is set to the number of folders in the `folderlist`.

Note
If the `messageFile` attribute is also specified, the attributed `messageFile` will be ignored.

Sample Rulefile

In the sample rulefile, the following files `libzz.so`, `libzz.sl`, and `libzz.dll` are library files. The */xx/yy* are directory paths that should be replaced by the relative paths to where these library files are located on the server.

```
#
# Sample rulefile
#
[General]
mailQuotaAttribute=mailquota
reportMethod=/xx/yy/libzz.so:myReportMethod [for Solaris only ]
/xx/yy/libzz.sl:myReportMethod [for HP-UX only]
\xx\yyiibzz.dll:myReportMethod [for Windows NT only]

[rule1]
notificationTriggerPercentage=60
enabled=1
notificationInterval=3
notificationMethod=/xx/yy/libzz.so:myNotifyMethod_60

[rule2]
notificationTriggerPercentage=80
enabled=1
notificationInterval=2
messageFile=/xx/yy/message.txt

[rule3]
notificationTriggerPercentage=90
enabled=1
notificationInterval=1
notificationMethod=/xx/yy/libzz.so:myNotifyMethod_90
#
# End
#
```

Threshold Notification Algorithm

1. Rule precedence is determined by increasing trigger percentages.
2. The highest applicable threshold is used to generate a notification. The time and the rule's threshold are recorded.
3. If users move into a higher threshold since their last quota notification, a new notification will be delivered based on the current set of applicable rules. This notice can be immediately delivered to any user whose space usage is steadily increasing.
4. If usage drops, the notification interval of the current rule (lower threshold) will be used to check the time elapsed since the last notice.
5. The stored time and threshold for the user will be reset to zero if the user's mailbox size falls below all of the defined thresholds.

Notification File

The utility depends on the message file to have at minimum a Subject Header. There should be at least one blank line separating the Subject from the body. The other required headers are generated by the utility. The notification file format is the following:

```
Subject: [Warning] quota reached for %U%


Hello %U%,
Your quota: %C%
Your current mailbox usage: %M%
Your mailbox is now %Q% full. The folders consuming the most space are:
%R%.

Please clean up unwanted diskspace.

Thanks,
-Administrator
```

where:

%U%	Specifies the user ID.
%Q%	Specifies the percentage of the mailbox quota used.
%R%	Specifies quota usage details, including assigned quota, total mailbox size, and percentage used. When -e is specified on the command line, the report shows the mailbox usage of the individual folders.
%M%	Specifies current mailbox size.
%C%	Specifies quota attribute value.

Note
If an account has less then 1KB of quota usage, imquotacheck prints out the usage in byte than kilobytes (KB) shown in the heading.

```
# ./imquotacheck -u testquota
Name Quota(K) Usage(K) % Quota# Usage# % OverDate WarnDate
-------------------- -------- -------- --- ------- ------- --- --------
testquota 256000 654B 0 100000 1 0 - -
-------------------------------------------------------------------------
```

Note
Localized versions of imquotacheck notification incorrectly convert the % and the $ signs. To correct the encoding, replace every $ with \24 and replace every % with \25 in the message file.

imsasm

imsasm

The `imsasm` utility is an external ASM (Application Specific Module) that handles the saving and recovering of user mailboxes. `imsasm` invokes the `imsbackup` and `imsrestore` 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 `imsbackup` or `imsrestore` command on the user's mailbox.

Location: *msg-svr-base*/`lib/msg`

Syntax

```
imsasm [<standard_asm_arguments>]
```

Options

The options used in the `imsasm` utility are also known as standard-asm-arguments, which are Legato NetWorker backup standards.

Either `-s` (saving), `-r` (recovering), or `-c` (comparing) must be specified and must precede any other options. When saving, at least one *path* argument must be specified. *path* may be either a directory or filename.

The following options are valid for all modes:

Option	Description
`-n`	Performs a dry run. When saving, walk the file system but don't attempt to open files and produce the save stream. When recovering or comparing, consume the input save stream and do basic sanity checks, but do not actually create any directories or files when recovering or do the work of comparing the actual file data.
`-v`	Turns on verbose mode. The current ASM, its arguments, and the file it is processing are displayed. When a filtering ASM operating in filtering mode (that is, processing another ASM's save stream) modifies the stream, its name, arguments, and the current file are displayed within square brackets.

When saving (`-s`), the following options may also be used:

Option	Description
`-b`	Produces a byte count. This option is like the `-n` option, but byte count mode will estimate the amount of data that would be produced instead of actually reading file data so it is faster but less accurate than the `-n` option. Byte count mode produces three numbers: the number of records, i.e., files and directories; the number of bytes of header information; and the approximate number of bytes of file data. Byte count mode does not produce a save stream so its output cannot be used as input to another asm in recover mode.
`-o`	Produces an “old style” save stream that can be handled by older NetWorker servers.
`-e`	Do not generate the final “end of save stream” Boolean. This flag should only be used when an ASM invokes an external ASM and as an optimization chooses not to consume the generated save stream itself.
`-i`	Ignores all save directives from `.nsr` directive files found in the directory tree.
`-f` *proto*	Specifies the location of a `.nsr` directive file to interpret before processing any files. Within the directive file specified by *proto*, *path* directives must resolve to files within the directory tree being processed, otherwise their subsequent directives will be ignored.
`-p` *ppath*	Prepends this string to each file’s name as it is output. This argument is used internally when one ASM executes another external ASM. *ppath* must be a properly formatted path which is either the current working directory or a trailing component of the current working directory.
`-t` *date*	The date after which files must have been modified before they will be saved.
`-x`	Crosses file system boundaries. Normally, file system boundaries are not crossed during walking.

When recovering (`-r`), the following options may also be used:

Option	Description
`-i` *response*	Specifies the initial default overwrite response. Only one letter may be used. When the name of the file being recovered conflicts with an existing file, the user is prompted for overwrite permission. The default response, selected by pressing `Return`, is displayed within square brackets. Unless otherwise specified with the `-i` option, `n` is the initial default overwrite response. Each time a response other than the default is selected, the new response becomes the default. When either `N`, `R`, or `Y` is specified, no prompting is done (except when auto-renaming files that already end with the rename suffix) and each subsequent conflict is resolved as if the corresponding lower case letter had been selected. The valid overwrite responses and their meanings are: • `n`—Do not recover the current file. • `N`—Do not recover any files with conflicting names. • `y`—Overwrite the existing file with the recovered file. • `Y`—Overwrite files with conflicting names. • `r`—Rename the conflicting file. A dot "." and a suffix are appended to the recovered file's name. If a conflict still exists, the user will be prompted again. • `R`—Automatically renames conflicting files by appending a dot "." and a suffix. If a conflicting file name already ends in a `.`*suffix*, the user will be prompted to avoid potential auto rename looping conditions.
`-m` *src*=*dst*	Maps the file names that will be created. Any files that start exactly with src will be mapped to have the path of dst replacing the leading src component of the path name. This option is useful if you wish to perform relocation of the recovered files that were saved using absolute path names into an alternate directory.
`-z` *suffix*	Specifies the suffix to append when renaming conflicting files. The default suffix is `R`.
path	Restricts the files being recovered. Only files with prefixes matching path will be recovered. This checking is performed before any potential name mapping is done with the `-m` option. When path is not specified, no checking is performed.

Examples

To use `imsasm` to save the mailbox `INBOX` for user `joe`, the system administrator creates a directory file *backup_root*`/backup/DEFAULT/joe/.nsr` with the following contents:

```
imsasm: INBOX
```

This causes the mailbox to be saved using `imsasm`. Executing the `mkbackupdir` utility will automatically create the `.nsr` file. See mkbackupdir.

imsbackup

imsbackup

The `imsbackup` utility is used to write selected contents of the message store to any serial device, including magnetic tape, a UNIX pipe, or a plain file. The backup or selected parts of the backup may later be recovered via the `imsrestore` utility. The `imsbackup` utility provides a basic backup facility similar to the UNIX `tar` command.

When `imsbackup`, `imsrestore`, `imsimport` or any processing intensive operation takes significantly more system resources than normal, and continues doing so longer than the `msprobe` interval, there may be a temporary backlog of DB transaction log files to be cleared. If there are more files than specified in `local.store.maxlog`, then `msprobe` may erroneously restart all the processes during a restore. To prevent this from happening, disable `msprobe` during the `imsbackup`, `imsrestore`, and `imsimport`.

Location: *msg-svr-base*`/sbin`

For more information about `imsbackup` and backing up the message store, see the Backing Up and Restoring the Message Store in the Messaging Server Administration Guide.

Syntax

```
imsbackup -f device [-b <blockfactor>] [-d <YYYYMMDD [:HHMMSS]>]
[-e <encoding>] [-u <file>] [-m <link_count>] [-ivlgrp] [ <name> ...]
```

Options

The options for this command are:

Option	Description
`-b` *blockfactor*	Everything written to the backup device is performed by blocks of the size 512 x blockfactor_. The default is `20`.
`-d` *YYYYMMDD [:HHMMSS]*	Date from which messages are to be backed up, expressed in YYYYMMDD [:HHMMSS]; for example, `-d 19990501:131000` would backup messages stored from May 1, 1999 at 1:10 pm to the present. The default is to back up all the messages regardless of their dates.
`-e` *encoding*	Mailbox name encoding (example: IMAP-mailbox-name)
`-f` *device*	Specifies the file name or device to which the backup is written. If `device` is "-', backup data is written to `stdout`.
`-g`	Debug mode. The output is written in the default log file not to the `stdout`. For the `stdout`, one should use `-v`.
`-i`	Ensures that all messages are backed up for a partial restore. Backs up every message multiple times if necessary.
`-l`	Used to autoload tape devices when end-of-tape is reached.
`-m` *link_count*	Specifies the minimum link count for hashing.
`-r`	Specifies read-only mode (does not upgrade or repair the mailbox automatically). This option is available starting in **Messaging Server 7 Update 4 Patch 18**.
`-u` *file*	Specifies a backup object file. This file contains the object names (entire message store, user, group, mailbox, and so on) to backup. See *name* for a list of backup object
`-v`	Executes the command in verbose mode.
`-x`	Backs up expunged messages. This option is available starting in **Messaging Server 7 Update 4 Patch 25**.
name	Can be 1) logical pathname of the backup object, 2) user ID, 3) message store mailbox name. Backup objects and paths: • Entire message store: `/` • Message store partition: */partition_name* (default: `/primary`) • Backup group---a group of users defined with regular expressions in a configuration file. See To Create Backup Groups for details. Path: */partition_name/backup_group* (`/primary/user` represents all users under `primary`). • User: */partition_name/backup_group/user_ID* • Mailbox: */partition_name/backup_group/user_ID/mailbox_name* • Message: */partition_name/backup_group/user_ID/mailbox_name/msgID* *user_IDs*: can be any user ID in the message store. If the user is not in the default domain, the user ID must be fully qualified (example: `Wally@siroe.com`). If user is in the default domain, the user ID can stand alone (example: `Wally`). *mailbox:* An email folder. It is specified using the following message store internal name: `user`/*user_ID*/*folder_name*. Note that `user` is a message store keyword.

Examples

The following example backs up the entire message store to `/dev/rmt/0`:

```
imsbackup -f /dev/rmt/0 /
```

The following backs up the mailboxes of user ID `joe` to `/dev/rmt/0`:

```
imsbackup -f /dev/rmt/0 /primary/user/joe
```

The following example backs up all the mailboxes of all the users defined in the backup group `groupA` to `backupfile:`

```
imsbackup -f- /primary/groupA > backupfile
```

imsconnutil

imsconnutil

Monitors user access of the message store. `imsconnutil` can provide the following information:

- Users currently logged in on IMAP or or any HTTP webmail client.
- The last access time (log in or log out) for a specified user.
- For IMAP: lists the authentication method, the IP address from which the users are logged in, the IP address to which users are connected, and the port on which they are logged to and from.

This command requires root access by the system user, and you must set the configuration variables `local.imap.enableuserlist`, `local.http.enableuserlist`, and `local.enablelastaccess` to 1.

See Monitoring User Access to the Message Store for more information and examples.

Location: *msg-svr-base*`/sbin`

Syntax

```
imsconnutil -c|-a [-s <service>] [-u <uid>] [-f <filename>]

imsconnutil -k {-u <uid> | -f <filename>}
```

Options

The options for this command are:

Option	Description
`-c|-a|-k`	At least one of `-c` or `-a`, or `-k` must be used.
`-a`	Last IMAP, POP, or http web mail client access (log in or log out) of user(s). `-s` does not affect the output of `-a`.
`-c`	List IMAPusers currently connected.
`-k`	Disconnect users from IMAP and POP. Users logged on to Communications Express lose the underlying IMAP connection and, thus, are also disconnected. The `-k` option must be used with either the `-u` *uid* option to specify users to disconnect or the `-f` *filename* option to specify the users listed in *filename*. The `-k` option requires that IMAP IDLE be configured. See Configuring IMAP IDLE for more information.
`-s` *service*	Can specify either imap or http as service to monitor. Only applies to `-c` option. POP is not available because POP users do not typically stay logged on.
`-u` *uid*	Specify a UID to monitor. If -u and -f are not listed, then all users are monitored.
`-f` *filename*	File containing UIDs to monitor. Each UID must be on its own line.

Examples

- List every user ID currently logged into IMAP and http.

imsconnutil -c

- List last IMAP, POP, or Messenger Express access (log in or log out) of every user ID.

imsconnutil -a

- List access history (last log off or log on) of all user IDs. Lists current user IDs logged into IMAP and http.

imsconnutil -a -c

- List IMAP users currently logged on the message store.

imsconnutil -c -s imap

- Reveal whether user ID George is logged onto IMAP or not.

imsconnutil -c -s imap -u George

- Reveal whether user ID George is currently logged onto IMAP or Messenger Express, and lists the last time George was logged o or off.

imsconnutil -c -a -u George

- Disconnect the user ID George.

imsconnutil -k -u George

Notes

The `rehostuser` utility makes use of the `imsconnutil` command. If you are planning to use the `rehostuser` command, for now, you must use the `configutil` command to explicitly configure `service.imap.ensidle=1`. This is because `imsconnutil` decides to use ENS or JMQ based on the configuration of the `service.imap.ensidle` parameter. While the `imapd` daemon treats that variable as "on" by default, if you have configured `ibiff`, `imsconnutil` treats the parameter as "off" by default. Starting with Messaging Server 7 Update 4, this situation is corrected.

imscripter

imscripter

The `imscripter` utility connects to an IMAP server and executes a command or a sequence of commands.

May be run remotely.

Location: *msg-svr-base*/`sbin`/

Syntax

```
imscripter [-h]
imscripter [-s <server>] [-p <port>] [-u <userid>] [-x <password>]
imscripter [-s <server>] [-p <port>] [-u <userid>] [-x <password>] -f
<scriptfile>
imscripter [-s <server>] [-p <port>] [-u <userid>] [-x <password>] -c
<command> [<command data>]
imscripter [-s <server>] [-p <port>] [-u <userid>] [-x <password>] -c
<command> -f <command data file>
```

Options

The options for this utility are:

Option	Description
`-c` *command*	Executes *command* - see discussion of commands below.
`-f` *file*	The *file* may contain one or more commands, or a list of mailboxes on which commands are to be executed. See discussion of commands below.
`-h`	Displays help for this command.
`-p` *port*	Connects to the given port. The default is `143`.
`-s` *server*	Connects to the given server. The default is `localhost`. The server can be either a host name or an IP address.
`-u` *userid*	Connects as *userid*.
`-v` *verbosity*	String containing options for printing various information. The options are as follows: E—Show errorsI—Show informational messages P—Show prompts C—Show input commands c—Show protocol commands B—Show BAD or NO untagged responses O—Show other untagged responses b—Show BAD or NO completion results o—Show OK completion results A—Show all of the above The letters designating options can be entered in any order. The default is `EPBibo`.
`-x` *passwd*	Uses this password.

Supported Commands

The imscripter command supports the following commands:

```
create <mailbox>
delete <mailbox>
rename <oldmailbox> <newmailbox< [<partition>]
getacl <mailbox>
setacl <mailbox> <userid> <rights>
deleteacl <mailbox> <userid>
```

If one or more of the above variables are included, the option executes the given command with that input. For example, `create lincoln` creates a mailbox for the user `lincoln`.

If the `-f` *file* option is used, the option executes the command on each variable listed in the file. For example:

```
# cat folders-to-create
xyz
abc
def
# imscripter -u <userid> -x <password> -c create -f folders-to-create
#
```

Raw IMAP commands

In addition to the commands which imscripter interprets (above), it will pass thru any raw IMAP command prefixed with an exclamation mark ("!"). For example:

```
# imscripter -u <userid> -x <password> -c \!list "" \*
   1) <= OK [CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS
CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE ESEARCH ESORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE CONTEXT=SEARCH CONTEXT=SORT
WITHIN SASL-IR SEARCHRES XSENDER X-NETSCAPE XSERVERINFO X-SUN-SORT
ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE X-SUN-IMAP X-ANNOTATEMORE XUM1 ID IDLE
AUTH=PLAIN] s4u-280rd-zone1-bur02.east.sun.com IMAP4 service (Oracle
Communications Messaging Exchange Server 7u4-20.01 64bit (built Nov 21 2010))
   1) <= OK User logged in
   2) <= CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS
CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE ESEARCH ESORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE CONTEXT=SEARCH CONTEXT=SORT
WITHIN SASL-IR SEARCHRES XSENDER X-NETSCAPE XSERVERINFO X-SUN-SORT
ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE X-SUN-IMAP X-ANNOTATEMORE XUM1 ID IDLE
   2) <= OK Completed
   2) <= LIST (\NoInferiors) "/" INBOX
   2) <= LIST (\HasNoChildren) "/" Drafts
   2) <= LIST (\HasNoChildren) "/" Sent
   2) <= LIST (\HasNoChildren) "/" Trash
   2) <= LIST (\HasNoChildren) "/" abc
   2) <= LIST (\HasNoChildren) "/" def
   2) <= LIST (\HasNoChildren) "/" test2
   2) <= LIST (\HasNoChildren) "/" xyz
   2) <= OK Completed
   3) <= BYE LOGOUT received
   3) <= OK Completed
#
```

Interactive Mode

If you issue the imscripter command without the -f and/or -c switches, it will go into interactive mode.
If you did not specify -u, it will prompt for userid.
If you did not specify -x, it will prompt for password.
You can then type imscripter commands or raw IMAP commands prefixed by "!" and see the response.
See discussion of commands above.

For example:

```
# imscripter
s4u-280rd-zone1-bur02 userid: <userid>
s4u-280rd-zone1-bur02 password for <userid>:
   1) <= OK [CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS
CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE ESEARCH ESORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE CONTEXT=SEARCH CONTEXT=SORT
WITHIN SASL-IR SEARCHRES XSENDER X-NETSCAPE XSERVERINFO X-SUN-SORT
ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE X-SUN-IMAP X-ANNOTATEMORE XUM1 ID IDLE
AUTH=PLAIN] s4u-280rd-zone1-bur02.east.sun.com IMAP4 service (Oracle
Communications Messaging Exchange Server 7u4-20.01 64bit (built Nov 21 2010))
   1) <= OK User logged in
   2) <= CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS
CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE ESEARCH ESORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE CONTEXT=SEARCH CONTEXT=SORT
WITHIN SASL-IR SEARCHRES XSENDER X-NETSCAPE XSERVERINFO X-SUN-SORT
ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE X-SUN-IMAP X-ANNOTATEMORE XUM1 ID IDLE
   2) <= OK Completed
imscripter> !select inbox
   3) <= FLAGS (\Answered \Flagged \Draft \Deleted \Seen $Forwarded)
   3) <= OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen
$Forwarded \*)]
   3) <= EXISTS
   3) <= RECENT
   3) <= OK [UNSEEN 14]
   3) <= OK [UIDVALIDITY 1266949413]
   3) <= OK [UIDNEXT 354]
   3) <= OK [READ-WRITE] Completed
imscripter> quit
   5) <= BYE LOGOUT received
   5) <= OK Completed
#
```

Individual Command Mode

As with interactive mode (above), -s and -p default to localhost and port 143 and imscripter will prompt for -u and -x if not specified.

With -c, you can execute an individual imscripter command and optionally have it operate on several folders. See the example of creating several folders in the discussion of commands above.

Script Mode

As with interactive mode (above), -s and -p default to localhost and port 143 and imscripter will prompt for -u and -x if not specified.

With -f, you can put imscripter and raw IMAP commands in a file and have imscripter execute that script. For example:

```
# cat script
create nmo
create blah
!list "" *
# imscripter -u <userid> -x <password> -f script
#
```

Notice the above displayed no output. It executed the command but did not display the output. Use the -v switch it specify which output you want to see. For example:

```
# imscripter -u <userid> -x <password> -f script -v A
Processing: script, verbosity: A, server: (<ask user>:143)
connecting to s4u-280rd-zone1-bur02:143...
addcallback CAPABILITY, NO, OK BAD
   1) => LOGIN <userid> *    *notice imscripter hid the password*
   1) <= OK [CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS
CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE ESEARCH ESORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE CONTEXT=SEARCH CONTEXT=SORT
WITHIN SASL-IR SEARCHRES XSENDER X-NETSCAPE XSERVERINFO X-SUN-SORT
ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE X-SUN-IMAP X-ANNOTATEMORE XUM1 ID IDLE
AUTH=PLAIN] s4u-280rd-zone1-bur02.east.sun.com IMAP4 service (Oracle
Communications Messaging Exchange Server 7u4-20.01 64bit (built Nov 21 2010))
   1) <= OK User logged in
   2) => CAPABILITY
   2) <= CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS
CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE ESEARCH ESORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE CONTEXT=SEARCH CONTEXT=SORT
WITHIN SASL-IR SEARCHRES XSENDER X-NETSCAPE XSERVERINFO X-SUN-SORT
ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE X-SUN-IMAP X-ANNOTATEMORE XUM1 ID IDLE
   2) <= OK Completed
   3) Cmd: create nmo
   3) => CREATE nmo
   3) <= NO Mailbox already exists        *this command failed because it
succeeded the first time*
   4) Cmd: create blah
   4) => CREATE blah
   4) <= NO Mailbox already exists        *ditto*
   5) Cmd: !list "" *
   5) => list "" *
   5) <= LIST (\NoInferiors) "/" INBOX
   5) <= LIST (\HasNoChildren) "/" Drafts
   5) <= LIST (\HasNoChildren) "/" Sent
   5) <= LIST (\HasNoChildren) "/" Trash
   5) <= LIST (\HasNoChildren) "/" abc
   5) <= LIST (\HasNoChildren) "/" blah
   5) <= LIST (\HasNoChildren) "/" def
   5) <= LIST (\HasNoChildren) "/" nmo
   5) <= LIST (\HasNoChildren) "/" test2
   5) <= LIST (\HasNoChildren) "/" xyz
   5) <= OK Completed
   7) => LOGOUT
   7) <= BYE LOGOUT received
   7) <= OK Completed
#
```

imsexport

imsexport

The `imsexport` utility exports Messaging Server folders into UNIX `/var/mail` format folders.

The `imsexport` utility extracts the messages in a message store folder or mailbox and writes the messages to a UNIX file under the directory specified by the administrator. The file name is the same name as the IMAP folder name. For message store folders that contain both messages and sub-folders, `imsexport` creates a directory with the folder name and a file with the folder name plus a `.msg` extension. The *folder*`.msg` file contains the messages in the folder. The *folder* directory contains the sub-folders.

Location: *msg-svr-base*`/sbin`

Syntax

```
imsexport -d <dir> -u <user> [-e <encoding>] [-g] [-s <mailbox>] [-v <mode>]
```

Options

The options for this command are:

Option	Description
`-d` *dir*	Specifies the destination directory name where the folders will be created and written. This is a required option.
`-e` *encoding*	Specify an encoding option.
`-g`	Specifies debugging mode.
`-s` *mailbox*	Specifies the source folder to export.
`-u` *user*	Specifies the message store id for a user. Note that this is not necessarily the login id of the user. The message store id is either *userid* (for default domain users) or *userid@domain* (for other users). This is a required option.
`-v` *mode*	Specifies verbose mode. The values for *mode* are `0`, `1`, and `2`. `0` specifies no output. `1` specifies mailbox level output. `2` (default) specifies message level output.

Example

In the following example, `imsexport` extracts all email for user `smith1`. `smith1` is a valid user account in the Sun Java System Messaging Server message store. User `smith1` has three folders on the store: `INBOX` (the normal default user folder), `private`, and `private/mom`. The destination directory will be `/tmp/joes_mail`.

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

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

- `/tmp/joes_mail/INBOX`
- `/tmp/joes_mail/private`
- `/tmp/joes_mail/private.msg`
- `/tmp/joes_mail/private/mom`

imsimport

imsimport

The `imsimport` utility migrates UNIX `/var/mail` format folders into a Messaging Server message store.

The `imsimport` utility extracts the messages stored in `/var/mail` mailboxes and appends them to the corresponding users' mailbox in the Messaging Server message store. Files in the directory that are not in the standard UNIX mailbox format are skipped. If the corresponding users do not exist in the message store, `imsimport` creates them. When the user quota is exceeded, `imsimport` bypasses the message store quota enforcement, so the user does not receive an "over quota" message.

The `imsimport` utility can be run while Messaging Server is running. If mail delivery is enabled for the mailbox you are importing, old mail can get mixed with new mail, so you might want to hold the delivery of this user during the migration. Mailbox access should not be a problem.

When `imsbackup`, `imsrestore`, `imsimport` or any processing intensive operation takes significantly more system resources than normal, and continues doing so longer than the `msprobe` interval, there may be a temporary backlog of DB transaction log files to be cleared. If there are more files than specified in `local.store.maxlog`, then `msprobe` may erroneously restart all the processes during a restore. To prevent this from happening, disable `msprobe` during the `imsbackup`, `imsrestore`, and `imsimport`.

> **Note**
> `imsimport` does not use the IMAP server. However, the `stored` utility must be running to maintain message store integrity. The LDAP server should be running if `imsimport` is expected to create new users.

Location: *msg-svr-base*/`sbin`/

Syntax

```
imsimport -u <user> -s _file_ [-c y|n] [-d <mailbox>] [-e <encoding>]
[-b] [-g] [-i] [-n] [-v <mode>]
```

Options

The options for this command are:

Option	Description
-b	Subscribes the destination mailbox folder when a new folder is created by the `imsimport` utility.
`-c y|n`	Provides an answer to the question: “Do you want to continue?” if an error occurs. Specify `y` for yes, `n` for no.
`-d` *mailbox*	Specifies the destination mailbox where the messages will be stored.
`-e` *encoding*	Specify an encoding option.
`-g`	Specifies debugging mode.
`-i`	Ignores the content-length field
`-n`	Creates a new mailbox with a *.date* extension if the mailbox exists. The *.date* extension is in the following form: *.mmddyy.HHMMSS* The month is specified by *mm*. The day is specified by *dd*. The year is specified by *yy*. For example, 052097 specifies May 20 in the year 1997. The time of day is specified by *HHMMSS*. For example 110000 specifies 11:00am.
`-s` *file*	Specifies the UNIX folder’s file name from which the messages are to be imported. The *file* parameter must be a full path name. This is a required option.
`-u` *user*	Specifies the message store identification for a user. Note that this should be identical in content and case to the UID attribute of the user's LDAP entry. If it is not the same, users will not be able to login to their INBOXES.
`-v` *mode*	Specifies verbose mode. The values for *mode* are `0`, `1`, and `2`. `0` specifies no output. `1` specifies mailbox level output. `2` (default) specifies message level output.

Examples

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

```
imsimport -u smith -s /var/mail/smith -d INBOX
```

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

```
imsimport -u smith -s /home/smith/folders/test -d test
```

If a destination folder called `test` already exists in the Messaging Server message store, `imsimport` appends the messages to the existing folder in the mailbox.

imsrestore

imsrestore

The `imsrestore` utility restores messages from the backup device into the message store. See Backing Up and Restoring the Message Store for more information.

When `imsbackup`, `imsrestore`, `imsimport` or any processing intensive operation takes significantly more system resources than normal, and continues doing so longer than the `msprobe` interval, there may be a temporary backlog of DB transaction log files to be cleared. If there are more files than specified in `local.store.maxlog`, then `msprobe` may erroneously restart all the processes during a restore. To prevent this from happening, disable `msprobe` during the `imsbackup`, `imsrestore`, and `imsimport`.

Location: *msg-svr-base*`/sbin`

Syntax

```
imsrestore -f <device>|- [-b <blocking_factor>] [-c y | n]
[-d <path>] [-D] [-e <encoding> [-h] [-i|-E] [-m <file>]
[-n] [-p] [-P partition] [-r <file>] [-s] [-S] [-t] [-u <file>] [-g] [-v
0|1|2|3|4|5]
[-x] [<name> . . .]
```

Options

The options for this command are:

Option	Description
`-b` *blocking_factor*	Indicates the blocking factor. Everything read on the device is performed by blocks of the size 512 x *blocking_factor*. The default is `20`. Note: this number needs to be the same blocking factor that was used for the backup.
`-c y | n`	Provides an answer to the question: "Do you want to continue?" if an error occurs. Specify `y` for yes, `n` for no.
`-d` *path*	restore all mailboxes under <path> (for multiple incremental restore)
`-e` *encoding*	Mailbox name encoding (example: IMAP-mailbox-name)
`-E`	Updates and expunges existing messages.
`-f` [*device*|*filename* |-]	Specify the device or file from which to read the backup data. Use `-f-` to read from `stdin` (ie, to pipe the data from some other command).
`-h`	Dumps the header.
`-g`	Debug mode
`-i`	Ignores existing messages. Does not check for existing messages before restore. Note that if you specify the `-i` option, you may have duplicate messages after the restore, since the `-i` option supersedes your ability to check for duplicates.

`-m` *file*	This mapping file is used when renaming user IDs. The format in the mapping file is *oldname=newname* with one set of names per line. For example:
	`a=x`
	`b=y`
	`c=z`
	where `a`, `b`, and `c` are old names and `x`, `y`, and `z` are new names. For users in a hosted domain, add `@domain_name` to the old and new user IDs.
	This option is used to rename user IDs when restoring from a Messaging Server 5.x, 6.x, or 7.x imsbackup file. Use the `-u` option when restoring from a SIMS 4.0 imbackup file.

`-n`	Creates a new mailbox with a `.date` extension (if the mailbox exists). By default, messages are appended to the existing mailbox.
`-p`	Restore mailboxes to original partition.
`-P`	Restore all mailboxes to this partition (ignore `-p` and `mailMessageStore`).
`-r` *file*	Reference file name (will restore all links in *file*).
`-s`	Used to restore a large file without using large file seeking.
`-t`	Prints a table of contents, but restore is not performed.

-u *file*	This file contains the object names (entire message store, user, group, mailbox, and so on) to restore. See *name* for a list of backup objects.
	For restoring SIMS 4.0 data into a Message Server 5.x, 6.x, or 7.x system, you can specify or rename users with `-u` *file*. Users should have one name per line. If you rename users, the format of *file* is `oldname=newname` with one set of names per line. For example:
	`joe`
	`bonnie`
	`jackie=jackie1`
	where `joe` and `bonnie` are restored, and `jackie` is restored and renamed to `jackie1`. Note that full object pathnames are not needed for user IDs.

`-v [0|1|2|3|4|5]`	Executes the command in verbose mode. `0`= no output `1`= output at mailbox level `2`= output at message level (default) `3`= print meta data (for use with `-t` only) `4`=print object level meta data (for use with `-t` only) `5`=print the backup data of mailboxes and messages (for use with `-t` only)
`-x`	Restores stubs into message files.
-S	
name	Can be 1) logical pathname of the backup object, 2) user ID, 3) *mailbox*. See imsbackup for description.

Examples

The following example restores the messages from the file `backupfile`:

```
imsrestore -f backupfile
```

The following example restores the messages for `joe` from the file `backupfile`:

```
imsrestore -f backupfile /primary/user/joe
```

The following example lists the content of the file `backupfile`:

```
imsrestore -f backupfile -t
```

The following example renames users in the file `mapfile`:

```
imsrestore -m mapfile -f backupfile
```

where the `mapfile` format is `oldname=newname`:

```
userA=user1
userB=user2
userC=user3
```

mboxutil

mboxutil

The `mboxutil` command lists, creates, deletes, renames, or moves mailboxes (folders). You can use the `mboxutil` command to report quota information, restore expunged messages, and list mailbox subscriptions.

> **Note**
> Do not "kill" the `mboxutil` process (`SIGKILL` (`kill -9`) command) in the middle of execution. Otherwise, you will need to restart the Messaging Server process.

You can use POSIX regular expressions in the `mboxutil` command.

Requirements: Must be run locally on the Messaging Server. The `stored` utility must also be running.

Location: *msg-svr-base*`/sbin`

Mailbox Naming Conventions

You must specify mailbox names in the following format:

`user`/*userid*/*mailbox*

where *userid* is the user that owns the mailbox and *mailbox* is the name of the mailbox.

For hosted domains, *userid* is *uid@domain.*

For example, the following command creates the mailbox named `INBOX` for the user whose user ID is `crowe`. `INBOX` is the default mailbox for mail delivered to the user `crowe`.

```
mboxutil -c user/crowe/INBOX
```

Important: The name `INBOX` is reserved for each user's default mailbox. `INBOX` is the only folder name that is case-insensitive. All other folder names are case-sensitive.

Syntax

```
mboxutil -l [-B <bound>] [-E <encoding>] [-p <MUTF7 IMAP pattern> | -P
<regular expression>] [-x | -s] [-D] [-z]

mboxutil -c [-E <encoding>] {<mailbox> | -f <file>}

mboxutil -C [-g num] [-E encoding] [-p <MUTF7 IMAP pattern> | -P <regular
expression>]

mboxutil -d [-E <encoding>] {<mailbox> | -p <MUTF7 IMAP pattern> | -P
<regular expression> | -f <file>}

mboxutil -R [-E <encoding>] <mailbox>

mboxutil -r [-E <encoding>] {<oldname> <newname> | -f <file> } [<partition>]

mboxutil -e [-E <encoding>] [-p <MUTF7 IMAP pattern> | -P <regular
expression> ]

mboxutil -o [-w <file> ] [-t <days>]

mboxutil -S [-n [-f <file>] | -u -f <file>]
```

Options

The options for this command are:

Option	Description
`-a`	Obsolete. Lists all user quota information. Use `imquotacheck`.
`-B` *bound*	Uses the user specified delimiter while listing the mailboxes. The option is always used with the `-l` option like mboxutil `-l [-B bound]`. The given bound can be any string.
`-c` *mailbox*	Creates the specified mailbox. A mailbox must exist before creating a secondary mailbox.
`-C`	Schedules cleanup immediately. Cleanup is automatically scheduled when mailboxes are expunged. This command can be used to schedule cleanup manually when: • The storage is very full. • `store.cleanupsize` has been reduced and you don't want to wait for users to do the next expunge. • Or you want to force cleanup on specific users/folders For more information see Mechanism to Schedule Cleanup Immediately.

`-d` *mailbox*	Deletes the specified mailbox. To delete a user from the message store, use the following value for `-d` *mailbox*: `user/`*userid*`/INBOX` For example, to delete the user `john` from the message store, use `-d user/john/INBOX`. To delete the `mm` folder in the user `john`'s mailbox, use `-d user/john/mm`. The recommended method to delete a user is to mark the user status as deleted in the LDAP directory (by using the Delegated Administrator utility `commadmin user delete` command or the Delegated Administrator console.). Next, remove resources from the user that have been marked as deleted for a period longer than a specified number of days by using the `msuserpurge` command for all mail services and `csclean` for all calendar services. Finally, use the `commadmin domain purge` command to permanently remove the users. For more information on removing users, see the following technical note: Deleting Messaging Server, Calendar Server, and Communications Express Users If you use the Delegated Administrator utility as described in the preceding paragraph, you do not have to use the `mboxutil -d` command to delete a mailbox. Note that `-p` and `-P` can be used in conjunction with one another.
`-D` This option is available starting in Messaging Server 7 Update 4 Patch 26.	Lists deleted mailboxes. When used with the `-l` option, lists deleted folders (as long as if the you enabled the `store.mailboxpurgedelay` option before the delete).
`-e`	Expunges all deleted messages in the message store. This option also can be used with the `-p` *pattern* or `-P` *regexp* options to expunge all deleted mailboxes with names that match *pattern* or *regexp*.
`-E` *encoding*	Specifies character set encoding used for the folder name display or the input of localized mailbox names. The default is that the folder name is displayed (or input accepted) in the localized character set (that is, in the form the user would expect to see). Using "`-E MUTF-7`" causes the folder name to be displayed (or input accepted) in the form used internally by the message store. For more information, see using Localized Mailbox Names in `imexpire`.
`-f` *file*	Specifies a file that stores mailbox names. The `-f` option can be used with the `-c`, `-r`, `-S`, or `-d` options. The file contains a list of mailboxes on which the `mboxutil` command is executed. The following is an example of entries in a data file: `user/daphne/INBOXuser/daphne/projxuser/daphne/mm`
`-l`	Lists all of the mailboxes on a server. `mboxutil -l` correctly displays characters associated with the system locale under which `mboxutil` is being executed. The `-P` *regexp* option accepts international characters.
-n	Lists personal, non-existing mailbox subscriptions. The `-n` option is used with the `-S` option.

`-o`	Checks for orphaned accounts. This option searches for inboxes in the current messaging server host that do not have corresponding entries in LDAP. For example, the `-o` option finds inboxes of owners who have been deleted from LDAP or moved to a different server host. For each orphaned account it finds, `mboxutil` writes the following command to the standard output: `mboxutil -d user/`*userid*`/INBOX` unless `-w` is specified
`-p` *pattern*	When used with the `-l` option, lists only those mailboxes with names that match *pattern*. Can also be used with the `-d` or `-e` option to delete or expunge mailboxes with names that match *pattern*. You can use IMAP wildcards. This option expects a pattern in IMAP M-UTF-7 format. This is not the recommended way to search for non-ASCII mailboxes. To search for non-ASCII mailboxes, use the `-P` option.
`-P` *regexp*	Lists, deletes, or expunges only those mailboxes with names that match the specified POSIX regular expression. This option expects the *regexp* in the local character encoding.
`-q` *domain*	Obsolete. Use `imquotacheck -d` *domain*
`-r` *oldname newname* [*partition*]	Renames the mailbox from *oldname* to *newname*. Optionally, to move a folder from one partition to another, specify the new partition with the partition option. This option can be used to rename a user. For example, `mboxutil -r user/user1/INBOX user/user2/INBOX` moves all mail and mailboxes from `user1` to `user2`, and new messages will appear in the new INBOX. (If `user2` already exists, this operation fails.) Folders are cleaned and purged automatically during the rename process. When used with `imsbackup` and `imsrestore`, you can use `mboxutil -r` to transfer ownership of a set of shared folders to another user while preserving the readership settings.
`-R` *mailbox*	Restores deleted messages that have not yet been purged. When a mailbox is expunged or expired, the uids of the deleted messages are stored in a `store.exp` file. The messages are physically removed by `impurge` depending on the values of `store.cleanupsize` and `store.cleanupage`. If expunge or expire is done by mistake, this option can be used to restore the deleted messages that have not yet been purged by `impurge` into the original mailbox.
`-s`	When used with the `-l` option, displays only the mailbox name. No other data is displayed.
-S	Lists personal mailbox subscriptions. When used with the `-n` option, the `-S` option lists or unsubscribes non-existing mailbox subscriptions.
-t *days*	When used with the `-o` option, lists the mailboxes that have not been accessed in a specified number of days (*days*). Thus, the `-t` option identifies inactive mailboxes (based on last-accessed date) together with orphaned mailboxes (mailboxes that do not have corresponding user entries in the LDAP directory). To identify (list) the orphaned and inactive mailboxes, use `mboxutil -o -w` *file* `-t` *num*. To delete these orphaned and inactive mailboxes, use `mboxutil -d -f` *file*, where *file* is the same file as the one passed to -w in the preceding command. To use this feature, the configuration variable `local.enablelastaccess` must be enabled for at least the number of days specified with the `-t` option.
`-u`	Unsubscribes personal non-existing mailbox subscriptions listed in the file specified with the `-f` *file* option.
`-w` *file*	Used with the `-o` option. Writes to a file the mailbox names generated by the `-o` option (which identifies orphaned accounts).

`-x`	When used with the `-l` option, displays the path and access control for a mailbox.
`-z`	When used with the `-l` option, displays a count of the mailboxes displayed.

Examples

To list all mailboxes for all users:

```
mboxutil -l
```

To list all mailboxes and also include path and ACL information:

```
mboxutil -l -x
```

To list all mailboxes displaying only the mailbox names:

```
mboxutil -l -s
```

To create the default mailbox named `INBOX` for the user `daphne`:

```
mboxutil -c user/daphne/INBOX
```

To delete a mail folder named `projx` for the user `delilah`:

```
mboxutil -d user/delilah/projx
```

To delete the default mailbox named `INBOX` and all mail folders for the user `druscilla`:

```
mboxutil -d user/druscilla/INBOX
```

To rename Desdemona's mail folder from `memos` to `memos-april`:

```
mboxutil -r user/desdemona/memos user/desdemona/memos-april
```

To schedule a cleanup of Dorothea's mailbox if there are at least 50 expunged messages:

```
$ mboxutil -C -g 50 -p user/dorothea/*
```

To restore messages that were expunged from Desdemona's `memos` mail folder in the last 24 hours:

```
mboxutil -R -t 24 user/desdemona/memos
```

To move the mail account for the user `dimitria` to a new partition:

```
mboxutil -r user/dimitria/INBOX user/dimitria/INBOX partition
```

where partition specifies the name of the new partition.

To move the mail folder named `personal` for the user `dimitria` to a new partition:

```
mboxutil -r user/dimitria/personal user/dimitria/personal partition
```

To list orphaned mailboxes and mailboxes that have not been accessed in 60 days:

```
mboxutil -o -w orphanfile -t 60
```

The preceding example writes the list of orphaned and inactive mailboxes to a file named `orphanfile`.

To delete orphaned and inactive mailboxes:

```
mboxutil -d -f orphanfile
```

where `orphanfile` is a file that has stored a list of orphaned and inactive mailboxes identified with the `-o` option.

To list personal, non-existing mailbox subscriptions for user mailboxes listed in a file named `orphanfile`:

```
mboxutil -S -n -f orphanfile
```

To unsubscribe the non-existing mailbox subscription list generated by the previous example:

```
mboxutil -S -u -f orphanfile
```

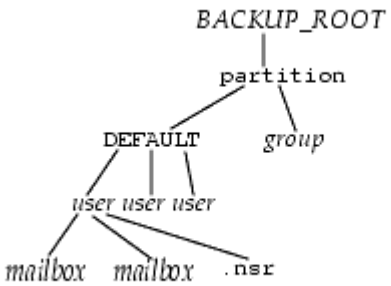
mkbackupdir

mkbackupdir

The `mkbackupdir` utility creates and synchronizes the backup directory with the information in the message store. It is used in conjunction with 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 new 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). The following figure displays the structure.

Backup directory hierarchy



Location: *msg-svr-base*`/sbin`

The variables in the backup directory contents are:

Variable	Description
BACKUP_ROOT	Message store administrator root directory.
partition	Store partition.
group	System administrator-defined directories containing user directories. Breaking your message store into groups of user directories allows you to do concurrent backups of groups of user mailboxes.To create groups automatically, specify your groups in the *msg-svr-base*`/config/backup-groups.conf` file. The format for specifying groups is:*groupname*= *patterngroupname* is the name of the directory under which the user and mailbox directories will be stored, and *pattern* is a folder name with IMAP wildcard characters specifying user directory names that will go under the *groupname* directory.
user	Name of the message store user.
folder	Name of the user mailbox.
mailbox	Name of the user mailbox.

The `mkbackupdir` utility creates:

- A default *group* directory (`ALL`) or the group directories defined in the `backup-groups.conf` configuration file. The following is a sample `backup-groups.conf` file:

```
groupA=a* (regexp)
groupB=b*
groupC=c*
.
.
.
```

- A *user* directory under the backup directory for each new user in the message store.
- A 0 length mailbox file for each mailbox.
- A `.nsr` file for each subdirectory that contains user mailboxes.

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 contains files of zero length. This includes the `INBOX`, which is located under the *user* directory.

> **Note**
> Make sure the backup directory is writable by the message store owner (mailsrv).

Syntax

```
mkbackupdir [-a <name_of_asm>] [-i | -f | -u] [-g] [-t <number_of_threads>]
[-e <encoding>] [-v] -p <directory>
```

Options

The options for this command are:

Option	Description
`-a` *name_of_asm*	Creates `.nsr` files using the specified asm name. This can be used for when you have multiple instances of Messaging Server as in symmetric HA environments.
`-e` *encoding*	Specify an encoding option.
`-f`	Backs up the folders only. By default, all mailboxes are backed up.
`-g`	Executes the command in debug mode.
`-i`	Backs up the inbox only. By default, all mailboxes are backed up.
`-p` *directory*	Specifies the directory for the backup image. This is a required option when `local.store.backupdir` is not configured.The max_thread parameter must be set between 1 and 1024.Note: The Networker has a limitation of 64 characters for *saveset* name. If your default backup directory pathname is too long, you should use this option to specify another pathname.
`-t` *number_of_threads*	Specifies the number of threads that can be used to create the backup directory. The default is one thread for each partition, which is usually adequate. If you have many partitions, and you do not want `mkbackupdir` to consume all your resources, you can lower this number.
`-u`	User level backup. Instead of backing up each folder as a file, create a backup file per user.
`-v`	Executes the command in verbose mode.

Examples

To create the `mybackupdir` directory, enter the following:

```
mkbackupdir -p /mybackupdir
```

moveuser

moveuser

> **Note**
> The MoveUser command has been deprecated. No future enhancements to MoveUser are planned. When moving mail between servers in a deployment, use the rehostuser command instead. The rehostuser command is easier to use and correctly handles more cases (such as deferring mail processing while the move is in progress). When moving mail between generic IMAP servers, there are a number of actively maintained open-source "imapcopy" utilities that are more flexible.
>
> In addition, MoveUser -l only supports Oracle LDAP Schema 1. If you have deployed Oracle LDAP Schema 2, use rehostuser instead.

The moveuser utility moves a user's account from one messaging server to another. When user accounts are moved from one messaging server to another, it is also necessary to move the user's mailboxes and the messages they contain from one server to the other. In addition to moving mailboxes from one server to another, MoveUser updates entries in the directory server to reflect the user's new mailhost name and message store partition.

May be run remotely.

Location: *msg-svr-base*/sbin/

> **Note**
> If you expect the MoveUser utility to alter the LDAP attributes, then you must run the following command to set the authentication cache timeout value to 0:
>
> ```
> configutil -o service.authcachettl -v 0
> ```
>
> MoveUser -l should be used for updating the directory. MoveUser -u does not alter the LDAP attributes.

Syntax

```
MoveUser -s <srcmailhost>[:<port>] -x <proxyuser>
 -p <password> -d <destmailhost<[:<port>]
[-u <uid> | -u <uid> -U <newuid>| -l <ldapURL>
 -D <binDN> -w <password> [-r <DCroot> -t
<defaultDomain>]] [-a <destproxyuser>] [-P <passwordfile]
```

Options

The options for this command are:

Option	Description

`-a` *destproxyuser*	ProxyAuth user for destination messaging server.
`-A`	Do not add an alternate email address to the LDAP entry.
`-d` *destmailhost*	Destination messaging server. By default, `MoveUser` assumes IMAP port `143`. To specify a different port, add a colon and the port number after *destmailhost*. For example, to specify port `150` for `myhost`, you would enter: `-d myhost:150`
`-D` *binddn*	Binding *dn* to the given *ldapURL*.
`-F`	Delete messages in source messaging server after successful move of mailbox. (If not specified, messages will be left in source messaging server.)
`-h`	Display help for this command.
`-i`	Uses the IMAP command `FETCH (RFC822)` instead of `FETCH (RFC822 HEADER RFC822.TEXT)` to workaround problems with another vendor's IMAP server.
`-l` *ldapURL*	URL to establish a connection with the Directory Server: `ldap://`*hostname*`:`*port*`/`*base_dn*`?`*attributes*`?`*scope*`?`_filter_For more information about specifying an LDAP URL, see your Directory Server documentation. Cannot be used with the `-u` option.
`-L`	Add a license for Messaging Server if not already set.
`-m` *destmaildrop*	Message store partition for destination messaging server. (If not specified, the default is used.)
`-n` *msgcount*	Number of messages to be moved at once.
`-o` *srcmaildrop*	Message store path for source messaging server. (If not specified, the default is used.)
`-P` *passwordfile*	Full path name of password file. The format of the password file is: ldapBind=*password* sourceProxyAdmin=*password* destProxyAdmin=*password*
`-p` *srcproxypasswd*	ProxyAuth password for source messaging server.
`-r` *DCroot*	DC root used with the `-l` option to move users under a hosted domain.
`-s` *srcmailhost*	Source messaging server.By default, `MoveUser` assumes IMAP port `143`. To specify a different port, add a colon and the port number after *srcmailhost*. For example, to specify port `150` for `myhost`, you would enter: `-s myhost:150`
`-S`	Do not set new message store partition for each user.
`-t` *defaultDomain*	Default domain used with the `-l` option to move users under a hosted domain.
`-u` *uid*	User ID for the user mailbox that is to be moved. Cannot be used with `-l` option. This option does not update the directory or modify LDAP attributes. Only the `-l` option will modify attributes.
`-U` *newuid*	New (renamed) user ID that the mailbox is to be moved to. Must be used with `-u` *uid*, where `-u` *uid*, identifies the old user name that is to be discontinued. Both the old and the new user ID must currently exist on both the source and the destination mailhost. After migration you are free to manually remove the original user ID from LDAP if you wish to do so.

`-v` *destproxypwd*	ProxyAuth password for destination messaging server.
`-w` *bindpasswd*	Binding password for the *binddn* given in the `-D` option.
`-x` *srcproxyuser*	ProxyAuth user for source messaging server.

The following options were introduced in **Messaging Server 7.0** to support source mail servers that do not support the IMAP `NAMESPACE` command. If any of these six options are specified they all must be specified.

Option	Description
`-I`	The mailbox prefix for private mailboxes. The meaning of the prefix is defined in RFC 2342.
`-J`	The mailbox prefix for user mailboxes. The meaning of the prefix is defined in RFC 2342.
`-K`	The mailbox prefix for public mailboxes. The meaning of the prefix is defined in RFC 2342.
`-X`	The mailbox delimiter for private mailboxes. The meaning of the delimiter is defined in RFC 2342.
`-Y`	The mailbox delimiter for user mailboxes. The meaning of the delimiter is defined in RFC 2342 .
`-Z`	The mailbox delimiter for public mailboxes. The meaning of the delimiter is defined in RFC 2342.

Examples

To move all users from `host1` to `host2`, based on account information in the Directory Server `siroe.com`:

```
MoveUser -l \
"ldap://siroe.com:389/o=siroe.com???(mailhost=host1.domain.com)" \
-D "cn=Directory Manager" -w password -s host1 -x admin \
-p password -d host2 -a admin -v password
```

To move one user from `host1` which uses port 150 to `host2`, based on account information in the Directory Server `siroe.com`:

```
MoveUser -l \
"ldap://siroe.com:389/o=siroe.com???(uid=userid)" \
-D "cn=Directory Manager" -w password -s host1:150 -x admin \
-p password -d host2 -a admin -v password
```

To move a group of users whose uid starts with letter "s' from `host1` to `host2`, based on account information in the Directory Server `server1.siroe.com`:

```
MoveUser -l \
"ldap://server1.siroe.com:389/o=siroe.com???(uid=s*)" \
-D "cn=Directory Manager" -w password -s host1 -x admin \
-p password -d host2 -a admin -v password
```

To move a user's mailboxes from `host1` to `host2` when the user ID of `admin` is specified in the command line:

```
MoveUser -l \
"ldap://server1.siroe.com:389/o=siroe.com???(uid=user1)" -s host1 -x admin \
-p password -d host2 -a admin -v password
```

To move a user named `aldonza` from `host1` to a new user ID named `dulcinea` on `host2`:

```
MoveUser -u aldonza -U dulcinea -s host1 -x admin -p password -d host2 -a
admin -v password
```

MoveUser can authenticate to the server as the administrator and use proxyauth to migrate user mailboxes. To migrate mailboxes from servers that do not support the proxyauth command, the admin can use the id of the migrating user as the admin id. Proxyauth is not performed when the admin id is the same as the user id. The `-x`, `-p`, `-a`, and `-v` options are not necessary for a proxyauth user.

To move a user named `joe` bypassing proxyauth:

```
MoveUser -u joe -s oldserver -x joe -p joepassword -d newserver -a admin -v
adminpassword
```

msuserpurge

msuserpurge

When user and domain mailboxes are marked for deletion, the `msuserpurge` command purges those user and domain mailboxes from the message store. Specifically, this command scans the following domain and user status attributes in LDAP for a value of deleted: `inetDomainStatus`, `mailDomainStatus`, `inetUserStatus`, `mailUserStatus`. This command can be run at the command line, or can be scheduled for execution with the `configutil` parameter `local.sched.userpurge`.

Requirements: If run manually, it must be manually run locally on the messaging server. Make sure that the environment variable `CONFIGROOT` is set to *msg-svr-base*`/config`.

Location: *msg-svr-base*`/lib`

Syntax

```
msuserpurge [-v|-r] [-d <domain>] [-g <days>]
```

Options

The options for this command are:

Option	Description
`-v`	Verbose (list deleted users).
`-r`	Report (list all users, do not purge).
`-g` *days*	Specify a grace period.
`-d` *domain*	Specify a domain.

Examples

```
msuserpurge -d siroe.com -g 0
```

readership

readership

The owner of an IMAP folder can grant permission for other users to access the folder. A folder that other users are allowed to access is called a shared folder. Users can modify the access rights on folders if their mail provides an interface to the `SETACL IMAP` commands. Administrators can use the `readership` utility to set or remove access rights on the folder, and to see how many users other than the owner are accessing a shared folder.

To list the rights on all shared folders, the mail administrator can use the `imcheck -d lright.db` command. To list rights on individual folders, use the `mboxutil -lxp` *folder* command.

Requirements: Must be run locally on the Messaging Server. The `stored` process must also be running.

Location: *msg-svr-base*`/sbin/`

Syntax

```
readership [ -d <days> |-p <months> ]
readership -r <folder pattern>
readership -s <foldername> <identifier> <acl_right>
readership -s -m <folder-pattern> <identifier> <acl_right>
readership [-e <encoding>] -s [ -m pattern | mailbox ] <identifier>
<acl_right>
```

To remove all rights for a user who has previously been granted access, set that user's rights to a null list, for example:

```
readership -s user/<userid>/<folder> <user-to-remove> ''
```

Options

The options for this command are:

Option	Description
`-d` *days*	Counts as a reader any identity that has selected the shared IMAP folder within the indicated number of days. The default is `30`.
`-p` *months*	Does not count users who have not selected the shared IMAP folder within the indicated number of months. The default is infinity and removes the seen flag data for those users. This option also removes the "seen" flag data for those users from the store.
`-r` *folder_pattern*	Lists the last access time of the folders matching the pattern.
`-s` *folder identifier acl_right*	Sets ACL rights character for folder, where *folder* is the name of the folder for which you are setting rights, *identifier* is the person or group to whom you are assigning the rights, and *acl_rights* are the rights you are assigning. See ACL Rights Characters. To remove a rights setting, specify a null set of of rights.
`-s -m` *folder_pattern identifier acl_right*	Sets ACL rights as `-s` but on all folders matching the specified pattern. Note: The `-m` *folder_pattern* option was added in Messaging Server 7 Update 4.
`-e` *encoding*	Specifies character set encoding used for the folder name display or the input of localized mailbox names. The default is that the folder name is displayed (or input accepted) in the localized character set (that is, in the form the user would expect to see). Using "`-e MUTF-7`" causes the folder name to be displayed (or input accepted) in the form used internally by the message store.

ACL Rights Characters

Character	Description
l	lookup - User can see and subscribe to the shared folder. (IMAP commands allowed: LIST and LSUB)
r	read - Users can read the shared folder. (IMAP commands allowed: SELECT, CHECK, FETCH, PARTIAL , SEARCH, COPY from the folder)
s	seen - Directs the system to keep seen information across sessions. (Set IMAP STORE SEEN flag)
w	write - Users can mark as read, and delete messages. (Set IMAP STORE flags, other than SEEN and DELETED)
i	insert - Users can copy and move email from one folder to another. (IMAP commands allowed: APPEND, COPY into folder)
p	post - Users can send mail to the shared folder email address. (No IMAP command needed)
k	create - Users can create new sub-folders. (IMAP command allowed: CREATE)
x	delete - Users can delete entries from the shared folder. (IMAP commands allowed: EXPUNGE, set STORE DELETED flag)
a	administer - Users have administrative privileges. (IMAP command allowed: SETACL)
t	delete - For messages, sets or clears \DELETED flag via STORE, or sets \DELETED flag during APPEND/COPY.
e	expunge - Performs EXPUNGE and expunge as a part of CLOSE.
n	access - Retrieves annotation information about the folder (see RFC 5257).

Example

Note: The `-m` *folder_pattern* was added in Messaging Server 7 Update 4.

```
# sbin/readership -s -m user/kellyc/a\* barbs lr
user/kellyc/abc: OK
user/kellyc/abc/123: OK
user/kellyc/abc/456: OK
```

reconstruct

reconstruct

The `reconstruct` utility rebuilds one or more mailboxes, or the master mailbox file (the mailboxes database), and repairs any inconsistencies. Use this utility to recover from almost any form of data corruption in the message store.

A mailbox consists of files under the user partition directory. The mailboxes database is the mboxlist database.

> Starting with **Messaging Server 7.0**, the `reconstruct` command has the additional options `-a` and `-t`.

Requirements: Must be run locally on the Messaging Server host; the `stored` utility must also be running.

Location: *msg-svr-base*/`sbin`/

Syntax

```
reconstruct [-n | -f] [-i] [-x] [-p partition] {-r [<mailbox...>] |
<mailbox...>}

reconstruct [-p <partition> [ -u <user> ] ] -m

reconstruct -a [-n | -f] [-t <nthreads>] [<mailbox...>]

reconstruct -A [-u userid]
reconstruct -l
reconstruct -q
reconstruct -s
```

Options

The options for this command are:

Option	Description
`-a` *mailbox...* Available starting with **Messaging Server 7.0**.	Addresses a user's mailbox comprehensively. If `reconstruct -a -f` is run, a full repair and reparse is forced. If `reconstruct -a -n` is run, only a consistency check is run, printing out errors found in the folder. This can be useful to catch issues before fixing them. Since there are many recovery functions embedded in the store libraries, running `-n` might still fix and modify the folder to some degree.
`-A` [-u *userid*]	Repairs the annotation database.
`-i`	Sets the `store.idx` file length to zero before reconstructing.

`-f`	Forces `reconstruct` to perform a fix on the mailbox or mailboxes.
`-x`	Recovers partially delivered messages.
`-l`	Reconstructs `lright.db`.
`-m`	Performs a consistency check and, if needed, repairs the mailboxes database. This option examines every mailbox it finds in the spool area, adding or removing entries from the mailbox's database as appropriate. The utility prints a message to the standard output file whenever it adds or removes an entry from the database. Specifically it fixes `folder.db`, `quota.db`, and `lright.db`

`-n`	Checks the message store only, without performing a fix on the mailbox or mailboxes. The `-n` option cannot be used by itself unless a mailbox name is provided. When a mailbox name is not provided, the `-n` option must be used with the `-r` option. The `-r` option may be combined with the `-p` option. For example, any of the following commands are valid:
	
	`reconstruct -n user/dulcinea/INBOX`
	
	`reconstruct -n -r`
	
	`reconstruct -n -r -p primary`
	
	`reconstruct -n -r user/dulcinea`

`-e`	Obsolete, see `mboxutil -R`.
`-o`	Obsolete, see `mboxutil -o`.
`-o -d` *filename*	Obsolete, see `mboxutil -o`.
`-p` *partition*	The `-p` option is used with the `-m` option and limits the scope of the reconstruction to the specified partition. If the `-p` option is not specified, `reconstruct` defaults to all partitions. Specifically it fixes `folder.db` and, `quota.db`, but not `lright.db`. This is because fixing the `lright.db` requires scanning the acls for every user in the message store. Performing this for every partition is not very efficient. To fix `lright.db` run `reconstruct -l`. Specify a partition name. Do not use a full path name.
`-q`	Fixes any inconsistencies in the quota subsystem, such as mailboxes with the wrong quota root or quota roots with the wrong quota usage reported. The `-q` option can be run while other server processes are running.
`-r` [*mailbox*]	Repairs and performs a consistency check of the partition area of the specified mailbox or mailboxes. The `-r` option also repairs all sub-mailboxes within the specified mailbox. If you specify `-r` with no mailbox argument, the utility repairs the spool areas of all mailboxes within the user partition directory.
`-s`	Repairs subscriptions.
`-t` *nthreads* Available starting with **Messaging Server 7.0**.	Performs a multithreaded reconstruct `-a` with *nthreads* threads (if `-t` is not specified, *nthreads* is chosen automatically, single-threaded if specific mailbox is specified).

`-u` *user*	The `-u` option is used with the `-m` option and limits the scope of the reconstruction to the specified user. The `-u` option must be used with the `-p` option. If the `-u` option is not specified, `reconstruct` defaults to all partitions or to the partition specified with the `-p` option. Specify a user name. Do not use a full path name.

The *mailbox* argument indicates the mailbox to be repaired. You can specify one or more mailboxes. Mailboxes are specified with names in the format `user/`*`userid`*`/`*`sub-mailbox`*, where *userid* is the user that owns the mailbox. For example, the inbox of the user `dulcinea` is entered as: `user/dulcinea/INBOX`.

Examples

The following command performs a reconstruct on a specific mailbox:

```
reconstruct user/dulcinea/INBOX
```

The following checks the specified mailbox, without performing a reconstruct:

```
reconstruct -n user/dulcinea/INBOX
```

The following command checks all mailboxes in the message store:

```
reconstruct -n -r
```

To Rebuild Mailboxes

To rebuild mailboxes, use the `-r` option. You should use this option when:

- Accessing a mailbox returns one of the following errors: "System I/O error" or "Mailbox has an invalid format".
- Accessing a mailbox causes the server to crash.
- Files have been added to or removed from the spool directory.

`reconstruct -r` first runs a consistency check. It reports any inconsistencies and rebuilds only if it detects any problems. Consequently, performance of the `reconstruct` utility is improved with this release.

You can use `reconstruct` as described in the following examples:

To rebuild the spool area for the mailboxes belonging to the user `daphne`, use the following command:

```
reconstruct -r user/daphne
```

To rebuild the spool area for all mailboxes listed in the mailbox database:

```
reconstruct -r
```

You must use this option with caution, however, because rebuilding the spool area for all mailboxes listed in the mailbox database can take a very long time for large message stores. (See reconstruct Performance.) A better method for failure recovery might be to use multiple disks for the store. If one disk goes down, the entire store does not. If a disk becomes corrupt, you need only rebuild a portion of the store by using the `-p` option as follows:

```
reconstruct -r -p subpartition
```

To rebuild mailboxes listed in the command-line argument only if they are in the `primary` partition:

```
reconstruct -p primary mbox1 mbox2 mbox3
```

If you do need to rebuild all mailboxes in the `primary` partition:

```
reconstruct -r -p primary
```

If you want to force reconstruct to rebuild a folder without performing a consistency check, use the `-f` option. For example, the following command forces a reconstruct of the user folder `daphne`:

```
reconstruct -f -r user/daphne
```

To check all mailboxes without fixing them, use the `-n` option as follows:

```
reconstruct -r -n
```

`reconstruct -r -f` focuses on fixing the folder index files. It assumes the folder record is good, and the peruser record is good. There are other commands to address these other data areas, such as reconstruct `-m`.

`reconstruct -a` attempts to address these all at once.

So if you think you need to repair an index file, but you're not sure if it is in the `folder.db`, you should not need to worry about running a `reconstruct -m` first, and whether the index corruption will be handled correctly there, or if both these will result in something that will conflict with the peruser entry later.

If you know your problem is with the index file, and there are no other complications, then you can go ahead and use `reconstruct -r -f` to save time.

Checking and Repairing Mailboxes

To perform a high-level consistency check and repair of the mailboxes database:

```
reconstruct -m
```

To perform a consistency check and repair of the primary partition:

```
reconstruct -p primary -m
```

> **Note**
> Running `reconstruct` with the `-P` and `-m` flags together will not fix `lright.db`. This is because fixing the `lright.db` requires scanning the ACLs for every user in the message store. Performing this for every partition is not very efficient. To fix the `lright.db` run `reconstruct -l`

To perform a consistency check and repair of an individual user's mailbox named `john`:

```
reconstruct -p primary -u john -m
```

You should use the `-m` option when:

- One or more directories were removed from the store spool area, so the mailbox database entries also need to be removed.
- One or more directories were restored to the store spool area, so the mailbox database entries also need to be added.
- The `stored -d` option is unable to make the database consistent.

  If the `stored -d` option is unable to make the database consistent, you should perform the following steps in the order indicated:
  - Shut down all servers.
  - Remove all files in *store_root*`/mboxlist`.
  - Restart the server processes.
  - Run `reconstruct -m` to build a new mailboxes database from the contents of the spool area.

`reconstruct` Performance

The time it takes `reconstruct` to perform an operation depends on the following factors:

- The kind of operation being performed and the options chosen
- Disk performance
- The number of folders when running `reconstruct -m`
- The number of messages when running `reconstruct -r`
- The overall size of the message store
- What other processes the system is running and how busy the system is
- Whether or not there is ongoing POP, IMAP, HTTP, or SMTP activity. Note that `reconstruct` is designed to run with the store services up. It is not necessary to keep the store offline to run `reconstruct`.

The `reconstruct -r` option performs an initial consistency check; this check improves `reconstruct` performance depending on how many folders must be rebuilt.

The following performance was found with a system with approximately 2400 users, a message store of 85GB, and concurrent POP, IMAP, or SMTP activity on the server:

- `reconstruct -m` took about 1 hour
- `reconstruct -r -f` took about 18 hours

> **Note**
> A `reconstruct` operation may take significantly less time if the server is not performing ongoing POP, IMAP, HTTP, or SMTP activity.

refresh

refresh

The refresh utility refreshes the configuration of the specified messaging server processes (SMTP, IMAP, POP, STORE, HTTP, ENS, SCHED). It is used when an option for one of the services has been modified and you wish this option to take effect.

Location: *msg-svr-base*/sbin

Syntax

```
refresh [ -h | <service>]
```

Options

The options for this command are:

Option	Description
-h	Show usage statement and list available services that can be refreshed.
service	Refresh just the specified service. Accepted values are watcher metermaid store imap pop cert http sched dispatcher job_controller mmp mta If no components are specified, all enabled services will be refreshed.

Examples

The following command refreshes the scheduler utility:

```
refresh sched
```

If refresh does not cause the change to take effect, then stop and restart the service.

rehostuser

rehostuser

The `rehostuser` utility enables you to move a Messaging Server user's mail store from one mailhost to another. It also disconnects any active session, locks the store to ensure atomicity of the move from the user's perspective (no loss of data, flag change, and so on), changes the user's LDAP entry, flushes LDAP caches as necessary, and causes any queued mail to be rerouted to the new store.

Requirements:
The following setup and configuration is required before you can use `rehostuser`:

1. Ensure that both the source and destination mailhosts are running Messaging Server 7.0 or later.
2. Ensure that both the source and destination mailhosts have IMAP IDLE configured.
3. If you are running Messaging Server prior to version 7 Update 4, also explicitly configure `service.imap.ensidle=1` by using the `configutil` command.
   This is because `imsconnutil` decides to use ENS or JMQ based on the configuration of the `service.imap.ensidle` parameter. While the `imapd` daemon treats that variable as "on" by default if you have configured `ibiff`, `imsconnutil` treats the parameter as "off" by default. Note that `service.imap.ensidle` is removed in Messaging Server 7 Update 4 and later and no longer needs to be set. `imsconnutil` always uses ENS.
4. Configure `ssh` on the source and destination mail hosts to allow the mail server user to perform a remote login.
   For more information, see Administering Messaging Server Remotely. Even if you start `rehostuser` as `root`, it still needs to run and execute `ssh` as the mail server user.
5. Make sure that `ugldapbinddn` has read and write access to the `mailHost`, `mailUserStatus`, and `mailMessageStore` attributes.
   New installations of Messaging Server 7.0, as well as upgrades to Messaging Server 7.0, should have this access made as part of the installation or upgrade process. The `rehostuser` utility checks for these permissions at startup and exits with an error if the permissions are insufficient.
6. The source mailhost must be capable of sending mail to the destination mailhost. In particular, the `tcp_intranet` channel on the source mailhost must be open for relaying, the mail must be routable to the new mailhost (directly or following `mx` records), and the destination mailhost must accept mail coming from the source mailhost.
7. `rehostuser` needs to consider LDAP replication delay. If either the source or destination Messaging Server system (or both) are configured to use an LDAP replica server instead of a master, there may be problems where attribute changes do not show up on the replica as quickly as required. This issue can be addressed as follows:
   a. If a Messaging Server points to a single Directory Server, configure the Directory Server in multi-master replication (MMR) mode so that it writes updates to its local database and replicates the change. (Note that MMR is the recommended way to setup Directory Server as opposed to the old slave/master model where slaves refer writes to the master, then have to wait for replication to get the updated data).
   b. If Directory Server failover capability is needed, use Directory Server proxy and enable one of the client affinity modes that guarantees that any single client will always see the updated data it just wrote.
   c. OpendDS 2.0, (possibly released 03/2009), may offer an *assured replication* feature where once a write is committed, it is guaranteed that the whole farm of replicated

Syntax

```
rehostuser -u userid -d mailhost [-p partition] [-c imsconnutil] [-b
imsbackup] [-s ssh] [-r imsrestore] [-e] [-o iss_src -t iss_dest -y scp_user
[-z src_path] [-w dest_path]]
```

Options

The options for this command are:

Option	Description	Default	Notes
`-u`	Specifies the user to move.		
`-d`	Specifies the fully qualifed domain name of the destination mailhost.		
`-p`	Specifies the destination partition.		
`-c`	Specifies the `imsconnutil` path	*msg-svr-base* `/bin`.	
`-b`	Specifies the `imsbackup` path	*msg-svr-base* `/bin`.	
`-s`	Specifies the `ssh` path	`/usr/bin/ssh`	Do not substitute `rsh` for `ssh`. The `rsh` command cannot pass failure status from the `imsrestore` command.
`-r`	Specifes the `imsrestore` path on the remote host.	*msg-svr-base* `/bin`	
`-e` This option is available starting with **Messaging Server 7 Update 4 Patch 25**.	Specifies extended availabilty of user's mailbox during move.		In this mode, Messaging Server performs a full backup and a remote restore without locking the user out of the system. Once the full backup and restore are complete, the user is locked out and Messaging Server performs an incremental backup and restore to take care of any changes that occurred since the start time of the full backup.
`-x` This option is available starting with **Messaging Server 7 Update 4 Patch 25**.	Moves the expunged message files.		

`-n` This option is available starting with **Messaging Server 7 Update 4 Patch 25**.	Does not remove the source mailbox.		
`-o` This option is available starting with **Messaging Server 7.0.5.30.0**.	Moves the ISS index from the specified source host.		
`-t` This option is available starting with **Messaging Server 7.0.5.30.0**.	Moves the ISS index to the specified destination host.		
`-z` This option is available starting with **Messaging Server 7.0.5.30.0**.	Specifies the ISS source installation path.	*iss-svr-base*	
`-w` This option is available starting with **Messaging Server 7.0.5.30.0**.	Specifies the ISS destination installation path.	Source installation path	
`-y` This option is available starting with **Messaging Server 7.0.5.30.0**.	ISS secure copy (scp) user name.		

rehostuser Example

This example shows how to move `user2` to mail server `bigdipper` where the Messaging Server software is installed in the `/opt/sun/comms/messaging` directory and the mail server user is `mailuser`.

```
# rehostuser -u user2 -d bigdipper.example.com -r
/opt/sun/comms/messaging/bin/imsrestore -s "/usr/bin/ssh -x -l mailuser"

disconnecting user2
-------------------------------------------
Tape Version : 2
Backup Date : 2008/01/08 17:45:16
Message Store : host1.example.com
Block factor : 20
-------------------------------------------
/second/user/user2/INBOX restoring...
/second/user/user2/INBOX/1 restoring...
/second/user/user2/INBOX/2 restoring...
/second/user/user2/INBOX/3 restoring...
/second/user/user2/folder20 restoring...
/second/user/user2/folder22 restoring...
/second/user/user2/folder33 restoring...
/second/user/user2/folder38 restoring...
/second/user/user2/folder49 restoring...
Mailbox user2 copied successfully.
Updated LDAP entry for uid=user2, ou=People, o=example.com, o=usergroup
Source mailbox deleted successfully.
```

relinker

relinker

`relinker` finds and relinks duplicate messages. Refer to Reducing Message Store Size Due to Duplicate Storage for more information.

Requirements: You may run `relinker` as `root` or `mailsrv`.

Location: *msg-svr-base*`/sbin`

Syntax

```
relinker [-p <partitionname>] [-d]
```

Options

The options for this command are:

Option	Description
`-d`	Specifies that the digest repository be deleted.
`-p` *partitionname*	Specifies the partition to be relinked. (default: all partitions).

Examples

To relink a message store:

```
# relinker


Processing partition: primary
Scanning digest repository...
Processing user directories...
-------------------------------------------------------
Partition statistics                Before        After
-------------------------------------------------------
Total messages                          73           73
Unique messages                         41           40
Message digests in repository            1            1
Space used                             55Kb         51Kb
Space savings from single-copy         40Kb         43Kb
-------------------------------------------------------

Note: run-time relinker (local.store.relinker.enabled) is not
enabled, so the repository will not be automatically purged.
When you're done with relinker, remember to purge the
repository by running "relinker -d"
```

After the "Scanning digest repository..." text shown above, relinker displays another '`.`' for every 100,000

messages message digests it finds in the repository and another '.' for every 100,000 messages as it is scanning messages. This indicates the progress of the `relinker` command.

To delete the digest repository:

```
# relinker -d


Processing partition: primary
Purging digest repository...
------------------------------------------------------------
Partition statistics                Before        After
------------------------------------------------------------
Message digests in repository            1            0
------------------------------------------------------------
```

Note that command-line relinker is also controlled by the local.store.relinker.maxage configutil option, which defaults to 24 (hours). To have command-line relinker consider all messages in the store, rather than just those delivered in the past day:

```
# configutil -o local.store.relinker.maxage
24
# configutil -o local.store.relinker.maxage -v -1
OK SET
# relinker


Processing partition: primary
Scanning digest repository...
Processing user directories...
------------------------------------------------------------
Partition statistics                Before        After
------------------------------------------------------------
Total messages                          75           75
Unique messages                         40           22
Message digests in repository            1           22
Space used                             51Kb         29Kb
Space savings from single-copy         50Kb         73Kb
------------------------------------------------------------

Note: run-time relinker (local.store.relinker.enabled) is not
enabled, so the repository will not be automatically purged.
When you're done with relinker, remember to purge the
repository by running "relinker -d"
# configutil -o local.store.relinker.maxage -v 24
OK SET
#
```

start-msg

start-msg

The `start-msg` utility starts all of the messaging server processes (`smtp`, `imap`, `pop`, `store`, `http`, `ens, sched`), or optionally, one specified service. Some services started by start-msg can be controlled by enabling or disabling the configutil parameters: `service.imap.enable`, `service.pop.enable`, `service.http.enable`, `local.smsgateway.enable`, `local.snmp.enable`, `local.imta.enable`, `local.mmp.enable`, `local.ens.enable`, and `local.sched.enable`. The ha option starts the server in HA mode. The watcher monitors process restarts processes on failure. The HA agent monitors the watcher process. In HA mode, the watcher will terminate when it gives up on restarting processes to trigger a failover.

Location: *msg-svr-base*/`sbin`

Syntax

```
start-msg [ -h | ha | <service>] [-m]
```

Options

The options for this command are:

Option	Description

`-h`	Shows the usage statement:
	`Usage: ./start-msg [COMPONENTS...]`
	`Start all or some Messaging Server components.`
	`Usage: ./start-msg ha`
	`Start all Messaging Server components in HA mode.`

`ha`	Starts all Messaging Server components in HA mode.

service	The accepted values for *service* are:
	`watcher`
	`mfagent`
	`ens`
	`metermaid`
	`store`
	`imap`
	`pop`
	`cert`
	`http`
	`sched`
	`dispatcher`
	`job_controller`
	`snmp`
	`sms`
	`mmp`
	`mta`
	If no components are specified, all enabled services will be started.

`-m`	Starts the message store as a replication master.

Examples

The following command starts all the Messaging Server processes:

```
start-msg
```

The following command starts the `imap` process:

```
start-msg imap
```

stop-msg

stop-msg

The `stop-msg` utility stops all Messaging Server processes (`smtp`, `imap`, `pop`, `store`, `http`, `ens`, `sched`), or optionally, one specified service. To use `stop-msg` *component*, the component must be enabled. The `stop-msg` command without arguments shuts down everything started by `start-msg`, including disabled components.

Location: *msg-svr-base*/`sbin`

Syntax

```
stop-msg [ -h | ha | [-f] <service>]
```

Options

The options for this command are:

Option	Description
`-f`	Force stop using `SIGKILL`.

`-h`	Shows the usage statement:
	`Usage: ./stop-msg [OPTIONS] [COMPONENTS...]`
	`  Stop all or some Messaging Server components.`
	`Usage: ./stop-msg ha`
	`  Stop all Messaging Server components in HA mode.`

`ha`	Stops all Messaging Server components in `HA` mode.

service	The accepted values for *service* are:
	`watcher`
	`mfagent`
	`ens`
	`metermaid`
	`store`
	`imap`
	`pop`
	`cert`
	`http`
	`sched`
	`dispatcher`
	`job_controller`
	`snmp`
	`sms`
	`mmp`
	`mta`
	If no components are specified, all enabled services will be stopped.

Examples

The following command stops all Messaging Server processes:

```
stop-msg
```

The following command stops the `http` service:

```
stop-msg http
```

stored

stored

The `stored` utility starts a daemon that performs the following functions:

- Performs checkpoint database transactions.
- Deadlock detection and rollback of deadlocked database transactions.
- Cleanup of temporary files and lock files on startup.
- Creates a database snapshot archive.
- Database recovery as necessary (see Message Store Automatic Recovery on Startup)

If any server daemon crashes, you must stop all daemons and restart all daemons including `stored`.

Requirements: Must be run locally on the Messaging Server.

Location: *msg-svr-base*`/lib/`

Syntax

To run `stored` as a daemon process:

```
stored -r | -R | -t [-v] | [-m] -d [site] | [-m]
```

Options

The options for this command are:

Option	Description
`-r`	Removes the database temporary files and synchronizes the database. This cleans up the database environment to prepare for an upgrade, downgrade, or migration.
`-R`	Removes the database temporary files without synchronizing the database. If `stored -r` fails because the software has been upgraded with an incompatible `libdb`, customers can run `stored -R` to remove the database temporary files. `stored -R` does not perform recovery.
`-t`	Checks the status of `stored`. The return code of this command indicates the status. To print the status, enter:`stored -t -v`
`-v`	Verbose output.
`-m`	Berkeley Database replication master.
`-d` *site*	Delete a mboxlist replication site. Site format is `host[:port]`. Default site is the local site.

imdbverify

imdbverify

> This command is available starting in **Messaging Server 7.0**.

The `imdbverify` utility takes and verifies message store database snapshots. For more information on running `imdbverify`, see Administering Message Store Database Snapshots (Backups).

Syntax

```
imdbverify -s [-m]
```

Options

The options for this command are:

Option	Description
-s	Takes a full snapshot of the message store database and saves it in the configured destination. Incremental snapshot is performed when this option is not specified.
-m	Verify all database files.

deliver

deliver

The `deliver` utility delivers mail directly to the message store.

If you are administering an integrated messaging environment, you can use this utility to deliver mail from another MTA, a `sendmail` MTA for example, to the Messaging Server message store.

> **Note**
> The `deliver` utility is only for use with files which are already completely and properly formed email messages (RFC 822).

Requirements: Must be run locally on the Messaging Server; the `stored` utility must also be running. Make sure that the environment variable `CONFIGROOT` is set to *msg-svr-base*`/config.`

Location on UNIX: *msg-svr-base*`/sbin/`

Syntax

```
deliver [-r <address>] [-m <mailbox>] [-g <flag>] [-q] [-c]
[-p] [-d] <userid>... | <pattern>
```

You can specify multiple userids. If you specify no options, mail is delivered to the inbox of the user specified in *userid*.

Options

The options for this command are:

Option	Description
`-c`	Automatically creates the mailbox if it doesn't exist in the message store. • The *mailbox* specified by the `-m` option will only be created if the sender specified by the *authid* passed in with the `-a` option has CREATE ("c") rights. • If no *mailbox* is specified, the inbox for *userid* will be created.
`-d`	This option is recognized but ignored by `deliver` in order to maintain compatibility with `/bin/mail`.
`-g` *flag*	Sets the system flag or keyword flag on the delivered message. • Valid system flags are `'\Seen','\Flagged','\Deleted','\Answered'` and `'\Draft'`. • The SEEN ("s"), DELETED ("d") and WRITE ("w") mailbox access control values determine whether the `'\Seen'`, `'\Deleted'` and all other flags can be set by the sender specified by the *authid* value that has been passed in with the `-a` option. • The value of *flag* will only be set if the sender has the required access rights to set that particular flag on the mailbox. • The `-g` option can be used multiple times to set more then one flag.
`-m` *mailbox*	Delivers mail to a specific *mailbox*. • Attempt to deliver the message to *mailbox* for each *userid*. If the access control on *mailbox* does not grant the POST ("p") right to the sender specified by the *authid* passed in with the `-a` option, the message will be delivered to the inbox for the *userid* instead. • If the *mailbox* does not exist for the *userid*, the message will be delivered to the inbox instead.
`-p`	This option was introduced in **Messaging Server 7 Update 1**. Deliver to the inbox of the users which match the wildcard *pattern*. • The `-c` and `-m` options have no impact when `-p` is used.
`-q`	Overrides mailbox quotas. Delivers messages even when the receiving mailbox is over quota.
`-r` *address*	Inserts a `Return-Path:` header containing address.
userid | *pattern*	Deliver to the mailbox of the user(s) specified by *userid*. Multiple *userid* values can be specified. *pattern* will be expanded to the appropriate list of existing users when the `-p` option is used.

Hint
The `mboxutil -lxp user/`*`userid/mailbox`* command can be used to check the access controls on *mailbox* for *userid*.

Debugging and Troubleshooting Options

Option	Description
`-t` *seconds*	Deliver as many copies of the message as possible to the *userid* in *seconds* time. This option is primarily meant for load-testing a mailbox.

Examples

To deliver the contents of a file named `message` to Fred's `tasks` mailbox:

```
deliver -m tasks fred < message
```

In the above example, if the `tasks` mailbox does not grant "p" rights to the value of the authid passed in with the -a option or the `tasks` mailbox does not exist, the contents of `message` is delivered to the inbox of the user `fred`.

To deliver the contents of a file named `message` to the inbox of all existing users on the store

```
deliver -p '*' < message
```

To deliver the contents of a file named `message` to the inbox of all existing users in the `siroe.com` hosted domain.

```
deliver -p '*@siroe.com' < message
```

Hint
Use the `mboxutil` command to determine which users will receive a copy of the message based on a given user *pattern*, e.g. `mboxutil -lp "user/`*`pattern`*`/INBOX"`

To deliver the contents of a file named `message` to Fred's `Important` mailbox, and set the `'\Flagged'` IMAP flag on that new message.

```
deliver -m Important -g '\Flagged' fred < message
```

impurge

impurge

The `impurge` command can be used to manually purge unused cache records and message files in mailboxes when the purge server daemon process is not running (local.purge.enable is disabled). It has the following options:

Requirements: Must be run locally on the Messaging Server.

Location: *msg-svr-base*`/lib/`

Syntax

```
impurge [-d] [-r hours] [-e]
```

Options

The options for this command are:

Option	Description
`-d`	Enables debug mode
`-r hours`	Exists after the specified number of hours
`-e`	Exits when the work queue is empty

If both `-r` and `-e` are specified, `impurge` exits when the queue is empty or time has expired.

Only one `impurge` process can be executed at a time. Attempting to run the `impurge` command whilst the purge server daemon is running will result in the following error message in the Messaging Server debug logs:

```
[24/Feb/2009:14:43:59 +1100] hostname impurge[17471]: General Error:
Could not get purge session lock.
Possibly another impurge is running
```

msgcert

msgcert

The `msgcert` utility should no longer be used and has been removed in **Messaging Server 7 Update 5** due to lack of support for modern security. Use `certutil` with appropriate options (`-Z sha256 -g 2048`), or other third-party certificate generation tools, to create certificates and certificate requests with up-to-date security strength.

Message Store Directory Layout

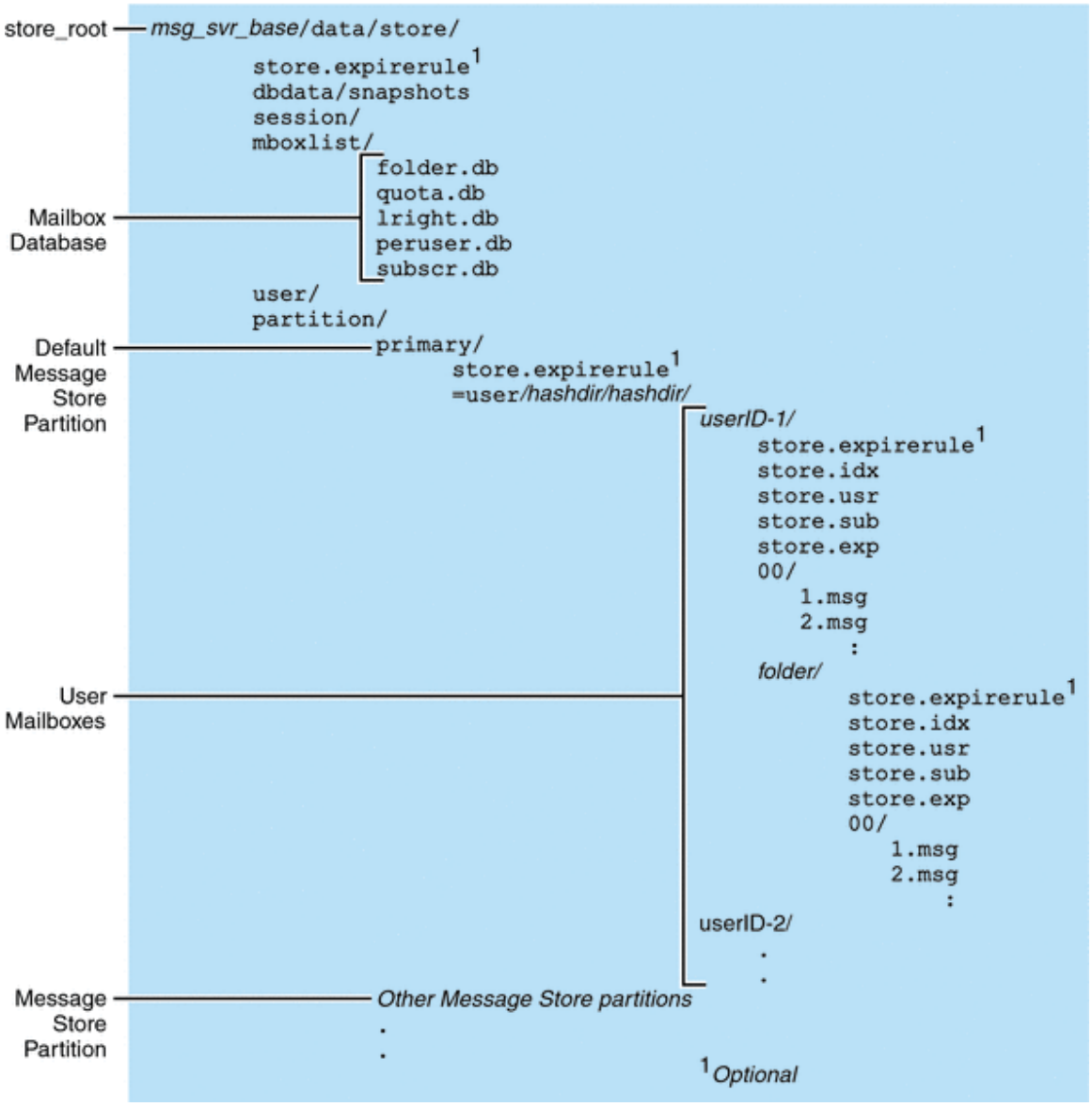
Message Store Directory Layout

The Message Store Directory Layout and Message Store Directory Description show and describe the message store directory layout.

Topic:

- Message Store Directory Layout

Message Store Directory Layout



The message store consists of a number of mailbox databases and the user mailboxes. The mailbox databases consists of information about users, mailboxes, partitions, quotas and other message store related data. The user mailboxes contain the user's messages and folders. Mailboxes are stored in a

message store partition, an area on a *disk* partition specifically devoted to storing the message store. See Managing Message Store Partitions and Adding Storage for details. Message store partitions are not the same as disk partitions, though for ease of maintenance, we recommend having one disk partition for each message store partition.

Mailboxes such as INBOX are located in the *store_root*. For example, a sample directory path might be:

```
<store_root>/partition/primary/=user/53/53/=mack1
```

The following table describes the message store directory.

Message Store Directory Description

Location	Content/Description
msg-svr-base	Default for Messaging Server 7 and greater: `/opt/sun/comms/messaging64` The directory on the Messaging Server machine that holds the server program, configuration, maintenance, and information files.
store_root	*msg-svr-base*`/data/store` Top-level directory of the message store. Contains the `mboxlist`, `user`, and `partition` subdirectories.
`./store.expirerule`	Contains the automatic message removal rules (expire rules). This optional file can be at different locations. See Message Store Message Expiration
store_root`/dbdata/snapshots`	Message store database backup snapshots that `stored` makes periodically.

store_root/mboxlist/	Contains mailbox database, database (Berkeley DB) that stores information about the mailboxes and quota information. annotate.db (introduced in Messaging Server 7) supports the ANNOTATE extension to IMAP, which permits clients and servers to maintain "meta data" for messages, or individual message parts, stored in a mailbox on the server. For example, you could use IMAP ANNOTATE to attach comments and other useful information to a message, or to attach annotations to specific parts of a message, marking them as seen or important, or a comment added. folder.db contains information about mailboxes, including the name of the partition where the mailbox is stored, the ACL, and a copy of some of the information in store.idx. There is one entry in folder.db per mailbox quota.db contains information about quotas and quota usage. There is one entry in quota.db per user. lright.db is an index for the folders by acl lookup rights. peruser.db contains information about per-user flags. The flags indicate whether a particular user has seen or deleted a message. subscr.db contains information about user subscriptions.
store_root/session/	Contains active message store process information.
store_root/user/	Not used.
store_root/partition/	Contains the message store partitions. A default primary partition is created. Place any other partitions you define in this directory.
store_root/partition/primary/=user/	Contains all the user mailboxes in the subdirectory of the partition. The mailboxes are stored in a hash structure for fast searching. To find the directory that contains a particular user's mailbox, use the hashdir utility.
.../=user/*hashdir*/*hashdir*/*userid*/	The top-level mail folder for the user whose ID is *userid*. This contains the user's INBOX. For the default domain, *userid* is *uid*. For hosted domains, *userid* is *uid@domain*. A user's incoming messages are delivered to the INBOX here.
.../*userid*/*folder*	A user-defined mailbox on the Messaging Server host.
.../*userid*/store.idx	An index that provides the following information about mail stored in the */userid/* directory: number of messages, disk quota used by this mailbox, the time the mailbox was last appended, message flags, variable-length information for each message including the headers and the MIME structure, and the size of each message. The index also includes a backup copy of mboxlist information for each user and a backup copy of quota information for each user.

`.../`*userid*`/store.usr`	Contains a list of users who have accessed the folder. For each user listed, contains information about the last time the user accessed the folder, the list of messages the user has seen, and the list of messages the user has deleted.
`.../`*userid*`/store.sub`	Contains information about user subscriptions.
`.../`*userid*`/store.exp`	Contains a list of message files that have been expunged, but not removed from disk. This file appears only if there are expunged messages.
`.../userid/ nn/ or` `.../userid/folder/nn/`	*nn* is a hash directory that contains messages in the format *message_id*`.msg`; *nn* can be a number from 00 to 99. *message_id* is also a number. Example: messages 1 through 99 are stored in the `.../00` directory. The first message is `1.msg`, the second is `2.msg`, third `3.msg`, and so on. Messages 100 through 199 are stored in the 01 directory; messages 9990 through 9999 are stored in the 99 directory; messages 10000 through 10099 are in the 00 directory, and so on.

Message Store Load Throttling

Handling Message Store Overload

> The following features described here were introduced in **Messaging Server 7.0**.

Topics:

Overview of Managing Message Store Load

An overloaded message store can suffer from degraded performance. The `mboxlist` database is particularly sensitive to overload conditions. When the database detects deadlocks, all database operations that cannot acquire the locks they need must abort the transactions and retry, thereby decreasing the throughput. If this situation continues, the message store can become very inefficient. In extreme cases, you need to restart the message store to recover.

Therefore, having the ability to control the message store load is crucial to prevent performance degradation. The message store uses transaction checkpoint time as the stress indicator. The `stored` daemon measures the transaction checkpoint duration (the time it takes to sync the database pages from the memory pool to disks). When the transaction checkpoint exceeds one minute, it raises an alarm and notifies the JES Monitoring Framework that it is stressed.

Message Store Load Throttling

Message store throttling is used to regulate short spikes of activities. When the `ims_master` program detects the stressed status from the message store, it informs the Job Controller. The Job Controller responds by temporarily decreasing the number of `ims_master` processes for the `ims-ms` channel. Similarly, when the LMTP server detects the stressed status, it tells the LMTP client, which informs the Job Controller, to back off. By decreasing the number of delivery threads, the Job Controller enables the message store to recover before performance begins to degrade.

Job Controller Stress Handling

Channel programs can now tell the Job Controller if they are being overwhelmed. If this occurs, then the job controller sees if it has happened recently. The job controller ignores stressed channel messages that are received within `StressBlackout` seconds of a previous stressed message for the same channel. If the message is processed, then the job controller will multiply the effective `threaddepth` parameter for the channel by `StressFactor`, and subtract `StressJob` from the job limit for the channel. `threaddepth` never goes over 134,217,727, and job limit never goes below 1. In addition, then job controller will ask all current master programs for the channel to exit, and will, if the queue is not empty, start an appropriate number of processes.

When `stressTime` seconds has passed after the last stress change, the job controller divides `threaddepth` by `UnstressFactor` (never allowing thread depth to drop below the original configured `threaddepth`), and adds `UnstressJob` to the job limit (never allowing the job limit to rise above the original configured limit. a "stress change" is either an increase in stress or a decrease in stress.

Job Controller Configuration

These configuration parameters go in the global section of the `job_controller.cnf` configuration file. The default values are:

```
StressBlackout=60
StressTime=120
StressFactor=5
StressJobs=2
UnstressFactor=StressFactor
UnstressJobs=StressJobs
```

Message Store Logging

Guide to Information on Using Message Store Log Messages

This page provides the following links to significant Message Store logging and error message pages:

- **Message Store Log Messages and Appropriate Actions.** Lists significant log messages and provides guidance in interpreting and addressing them.
- **Overview of Messaging Server Logging.** Provides an introduction to Messaging Server logging concepts.
- **Tools for Managing Logs.** Lists available tools for managing logs.
- **`configutil` Logging Parameters.** Search the parameters for the string *log*.
- **General Discussion on Message Store Logging.** Describes logging for the Message Store (POP, IMAP, and HTTP), Admin, and Default services.

Message Store Maintenance Queue

Message Store Maintenance Queue

> The message store maintenance queue was introduced in **Messaging Server 7.0**.

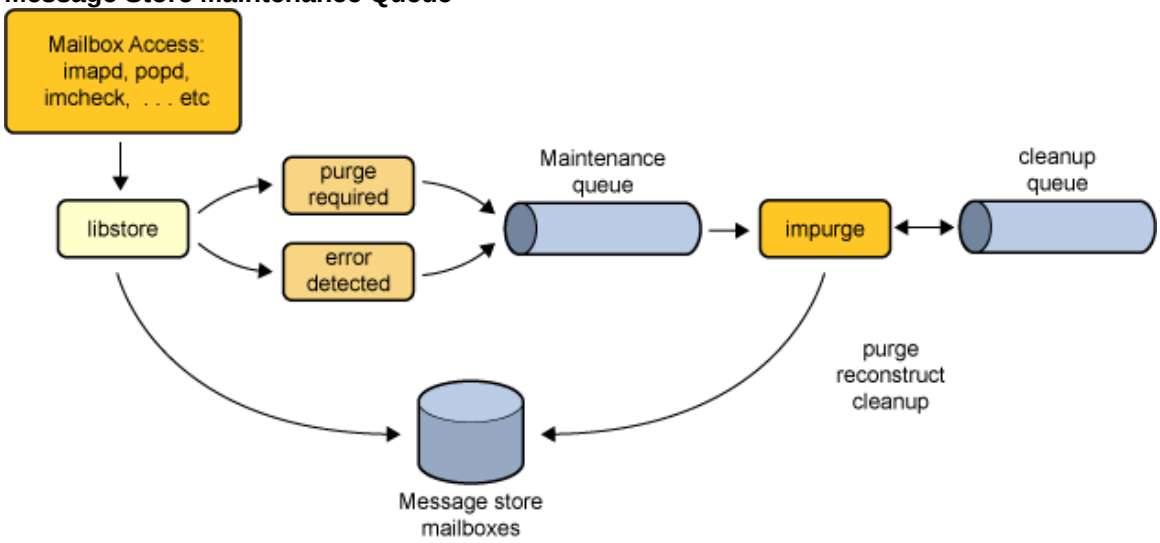
In addition to support for very large mailboxes , the way in which the message store purges mailboxes has changed starting in Messaging Server 7.0, making the process more efficient when dealing with a large mailbox. Beginning with Messaging Server 7.0, only the index file is purged when a mailbox is expunged. The purging of cache records is deferred until the amount of expunged data has exceeded a configurable limit. In addition, the message store uses a maintenance queue to schedule mailbox purge and repair tasks. Mailbox corruptions detected by the message store are also queued for repair automatically.

Topics:

Overview of Message Store Maintenance Queue

The following diagram summarizes the major components and their interactions with the queue. See also this diagram of a mailbox structure.

Message Store Maintenance Queue



When a mailbox is expunged, the message store processes update the number of expunged messages and enqueue the mailbox name for purging when the number of expunged messages has exceeded the configured limit (set by the `store.cleanupsize` parameter). Similarly, when store corruptions are detected by the store processes, they enqueue the mailbox name for repair. A maintenance queue entry contains a mailbox name, a task ID, and a timestamp. The `impurge` process dequeues the entries, checks the timestamp and mailbox size or last repair time and performs the maintenance task if required.

The mailbox size and last repair time are stored in the `mboxlist` database. Mailboxes that have already been repaired after the enqueue time are ignored. Obsoleted cache and message files are enqueued for removal after the cleanup age has expired.

You can set the `impurge` process to execute as a daemon or through the scheduler. When you use the the scheduler, you can configure `impurge` to exit when the queue is empty or when the end time expires. In this way, you can configure `impurge` to run during off peak hours.

`impurge` executes as a daemon by default.

Displaying the Maintenance Queue

Use the `imcheck -q` command to display the contents of the maintenance queue, for example:

```
./imcheck -q
RecNo      Mailbox                              Timestamp      Action
---------- ------------------------------------ -------------- ------
2          user/username                        20080225095558 Clean
```

The maintenance queue database is stored under the *msg-svr-base*`/data/store/mailbox/mqueue` directory and persists across Messaging Server restarts.

Deleting, Expunging, Purging, and Cleaning up Messages

Message removal is a four steps process:

1. Delete
   - A user deletes a message. This results in the per-user `\Deleted` flag being set on the message. If there is a second client, the deleted flag may not be recognized immediately by that second client. You can set the `local.imap.immediateflagupdate` parameter to enable immediate flag update.
2. Expunge and Expire
   - When the mailbox is expunged, messages with the `\Deleted` flag are removed from the `store.idx` file. The message itself is still on disk, but once messages are expunged, clients can no longer restore them. The number of the expunged messages is also recorded in the "Expunged" mailbox meta-data field. This field can be reviewed using the `./imcheck -m` command.
   - When a message matches an expiration rule during an `imexpire` run, the `imexpire` command removes the message from the `store.idx` file and increments the "Expunged" mailbox meta-data field.
3. Cleanup
   - Cleanup of a folder is scheduled when the number of expunged in the folder messages exceeds `store.cleanupsize`, default is 100. The `store.exp` file is renamed to `store.exp.timestamp`.
   - Obsolete message files and cache files are removed from the store partitions when the `store.cleanupage` period has elapsed. `store.cleanupage` controls the cleanup grace period.
4. Purge
   - `impurge` purges the cache files when the number of expunged cache records exceeds `store.purge.count` (default is 500) and the percentage of expunged cache records exceeds `store.purge.percentage` (default is 5%). Note that this is different from the `imsimta purge` command and `local.schedule.purge` parameter, which purges older versions of the MTA log files.

Mailbox Self Healing (Auto Repair)

The message store performs sanity checks when a mailbox is opened and when index and cache data are accessed. When the message store encounters a mailbox format error, and error is returned to the client and the mailbox is enqueued for repair. `impurge` dequeues the mailbox and repairs the mailbox automatically. The mailbox size and last repair time are stored in the mboxlist database. Mailboxes that have already been repaired after the enqueue time are ignored.

The following is an example of the auto repair process. In this example, the `store.idx` file for a user's INBOX has been removed. When the user accessed their INBOX the following error was reported in the `imap` log file:

```
[25/Feb/2008:09:55:57 +1100] hostname imapd[7264]: Store Critical:
Unable to open mailbox user/username: Mailbox does not exist
[25/Feb/2008:09:55:57 +1100] hostname imapd[7264]: Store Error:
user/username: Mailbox has an invalid format; repair requested
```

In the `default` log file an impurge record indicates that the requested mailbox repair as been processed and the store.idx restored:

```
[25/Feb/2008:09:55:58 +1100] hostname impurge[7263]: General Notice:
repairing user/username
```

Maintenance Queue configutil Parameters

The `local.purge.enable` parameter enables the purge server daemon process on `start-msg` startup. Various other `store.*` parameters control the maintenance queue. For more information on these parameters, see the Configutil Reference.

The impurge Command

You can use the `impurge` command to manually purge unused cache records and message files in mailboxes when the purge server daemon process is not running, that is, when the `local.purge.enable` parameter is disabled.

Attempting to run the impurge command while the purge server daemon process is running results in the following error message in the Messaging Server `default` log:

```
[24/Feb/2009:14:47:15 +1100] hostname impurge[17986]: General Error:
Could not get purge session lock.
Possibly another impurge is running
```

For details, see impurge.

Message Store Message Expiration

Message Expiration Overview

This information describes message expiration concepts. See Configuring Message Expiration for information on message removal tasks.

Message expiration automatically removes messages from the message store based on criteria that you set. For example, you can remove old messages, overly large messages, seen or deleted messages, messages with specific `Subject:` lines, messages of a certain type, and so on.

> **Note**
> Messaging Server removes messages without giving a warning, so it is important to inform users about message expiration policies. Unexpected message removal can be a source of consternation for users and administrators.

Topics:

imexpire Theory of Operation

Message expiration is performed by the `imexpire` utility, which performs a specific action to the expired messages. (See How the Message Store Removes Messages for details on the message removal process.) You can launch the `imexpire` utility from the command-line or schedule it to launch through the `imsched` daemon. You specify a set of expiration rules in the `store.expirerule` file. You can have multiple rules files, each located the directory that pertains to the scope of the rules. That is, rules that apply globally to the entire message store are put in one directory, rules that apply to a partition in another, rules that apply to users in yet another, and so on.

> **Note**
> Although global expiration rules can be specified with `store.expire.`*attribute* `configutil` options, use `store.expirerule` files to specify these rules. If too many rules are created by using `configutil`, performance problems can result.

`imexpire` loads all of the expire rules at start up. By default, `imexpire` creates one thread per partition. Each thread goes through the list of user folders under its assigned partition and loads the local expire rule files as it goes. The expire function checks each folder against the expire rules applicable to this folder and expunges messages as needed.

It is also possible to exclude specified users from the expire rules by adding their user ID, one per line, in a file called `expire_exclude_list` in the *msg-svr-base*`/config/` directory.

To Deploy the Message Expiration Feature

Message expiration requires the following three steps:

1. Define message expiration policy: Which messages will be expired? What users, folders, domains, and partitions will have messages expired? What size, message age, and headers will define the

removal criteria? Define the scope of messages to be removed. See To Define Automatic Message Removal Policy.
2. Specify the `imexpire` rules to implement this policy. See To Set Rules Implementing Automatic Message Removal Policy.
3. Specify the `imexpire` scheduling. See To Schedule Automatic Message Removal and Logging Level.

To Define Message Expiration Policy

You can define message expiration based on criteria such as:

- **Age of Message** – Expire messages older than *X* days. Attribute: `messagedays`.
- **Message Count** – Expire messages in a folder exceeding X messages. Attribute: `messagecount`.
- **Age of Oversized Message** – Expire messages that exceed X bytes after Y days grace period. Attributes: `messagesize` and `messagesizedays`.
- **Seen and Deleted Message Flag** – Expire messages with the *Seen* or *Deleted* flag set. These criteria can be set to "`and`" or "`or`." If set to `or`, the message's Seen/Delete flag will cause expiration regardless of other criteria. If set to `and`, the message's Seen/Delete flag must be set along with passing all other specified criteria. Attributes: `seen` and `deleted`.
- **Header Field of Message** – Allows you to specify a header and string as criteria for expiring a message, for example, removing all messages with the header "Subject: Work from Home." Note that this feature also allows you to use message type as a criteria too. See Expiring Messages by Message Type.
- **Folder of Messages** – Allows you to specify the folder on which to expire messages. Attribute: `folderpattern`. Note that this attribute only uses the modified UTF-7 character set.

> **Note**
> `imexpire` does not allow you to delete or preserve messages based on how long it has been since that message was read. For example, you cannot specify that messages that have not been read for 200 days will be removed.

Examples of Message Expiration Policy

Example 1: Remove all messages 365 days old in a folder exceeding 1,000 messages.

Example 2: Remove messages in domain `siroe.com` that are older than 180 days.

Example 3: Remove all messages that have been marked as **deleted**.

Example 4: Remove messages in sesta.com that have been marked as **seen**, are older than 30 days, are larger than 100 kilobytes, from folders exceeding 1,000 messages, with the header `X-spam.`

To Set Rules Implementing Message Expiration Policy

Rules are set by putting them into a `store.expirerule` file. An example of two global `store.expirerule` rules is shown below:

```
Rule1.regexp: 1
Rule1.folderpattern: user/.*/trash
Rule1.messagedays: 2
Rule2.regexp: 1
Rule2.folderpattern: user/.*
Rule2.messagedays: 14
```

In this example, Rule 1 specifies that all messages in all users' trash folders are removed after two days. Rule 2 specifies that all messages in any folder in the message store are removed after 14 days.

This section consists of the following subsections:

Expiration Rules Guidelines

This section sets the guidelines for the `store.expirerule` file rules.

> **Note**
> In earlier Messaging Server releases, expiration rules could be set with `configutil` parameters `store.expirerule.`*attribute* (see http://msg.wikidoc.info/index.php/Configutil_Reference.) This is still true, but expire rules using header constraints (example: expiring a message with a specific subject line) are not supported. Also, regular expressions in the expire rules created with `configutil` need to be POSIX compliant rules. If you want to use UNIX compliant regular expressions you must use the `store.expire` file. In addition, using both `configutil` options and the global `store.expirerule` configuration file is not supported. If the configuration file is present, `configutil` options are not used. In any case, it is best to use `store.expirerule` to specify all expiration rules.

- Rules are specified in a file called `store.expirerule`.
- Multiple expiration criteria can be specified with the same rule. (See preceding example.)
- Rules can apply to the entire message store (global rules), a partition, a user, or a folder.
  - The global rules are stored in the *msg-svr-base*`/config/store.expirerule` file.

    > **Note**
    > Each global rule will be checked against every mailbox, which can cause some processing overhead depending on the number of global rules you specify. For this reason, you should not specify partition, mailbox or user rules in the global rules file. In general, you should try not to put any more expiration rules than necessary in this file.

  - Partition rules are stored in *store_root*`/partition/`*partition_name*`/store.expirerule` (more accurately, the location specified by the `store.partition.*.path configutil` option).
  - User rules are specified in *store_root*`/partition/`*partition_name*`/userid/store.expirerule` or by specifying the `folderpattern` rule to be `user/`*userid*`/.*`.
  - Folder rules are specified in *store_root*`/partition/`*partition_name*`/userid/folder/store.expirerule` or by specifying the `folderpattern` rule to be `user/`*userid*`/`*folder*.

    Multiple non-global rules (user, folder, partition) using *rule_name* was only implemented starting in **Messaging Server 6.2p4**.

- Multiple expire rules can be applied to a mailbox at the same time. An expire policy for a mailbox consists of global rules and local rules. Local rules apply to the mailbox under the same directory and all of its sub-folders.
- `imexpire` unifies all of the expiration rules applying to a mailbox, unless there is an exclusive rule specified for this mailbox (see `imexpire` Attributes). The resulting rule set represents the most restrictive expiration policy based on all applicable rules. For example, if rule X expires messages

such that the maximum message life is 10 days, and rule Y specifies 5 days, the union will be 5 days.

`imexpire` Attributes

Attribute	Description (Attribute Value)
`action`	Specifies an action to perform on the messages caught by the expire rules. The possible values are: • `discard` – discards the message. This is the default. • `report` – prints the mailbox name, uid-validity and uid to stdout. • `archive` – archives the message with the Compliance and Content Management System and then discards the message. • `fileinto:` *folder* – files the message into the specified folder. The shared folder prefix can be used to file messages to folders owned by another user.
`exclusive`	Specifies whether or not this is an exclusive rule. If specified as `exclusive`, only this rule applies to the specified mailbox(es) and all other rules are ignored. If more than one exclusive rule exists, the last exclusive rule loaded will be used. For example, if a global and a local exclusive rule are specified, the local rule will be used. If there is more than one global exclusive rule, the last global rule listed by `configutil` is used. (`1`/`0`)
`expires`	`imexpire` will select the message if the date value specified with these header fields is older than the expiration date based on the messagedays attribute. If multiple expiration header fields are specified, the earliest expiration date will be used. (string).
`expiry-date`	`imexpire` will select the message if the date value specified with these header fields is older than the expiration date based on the messagedays attribute. If multiple expiration header fields are specified, the earliest expiration date will be used. (string).
`folderpattern`	Specifies the folders affected by this rule. The format must start with a `user/`, which represents the directory *store_root*`/partition/*/`. See `imexpire` Folder Patterns Using Regular Expressions. (POSIX regular expression)
`messagecount`	Maximum number of messages in a folder. Oldest messages are expunged as additional messages are delivered. (integer)
`foldersize`	Maximum size of folder before the oldest messages are expunged when additional messages are delivered. (integer in bytes)
`messagedays`	Number of days in the message store before being expunged. (integer)
`messagesize`	Maximum size of message in bytes before it is marked to be expunged. (integer)
`messagesizedays`	Grace period. Days an over-sized message should remain in a folder. (integer)
`messageheader.` *header*	Specifies a header field and string. Values are not case-sensitive and regular expressions are not recognized. Example: `Rule1.messageheader.Subject: Get Rich Now!`
`regexp`	Enable UNIX regular expressions in rules creation. (1 or 0). If not specified, IMAP expressions will be used.

savedays	Number of days the messages are saved in a folder until they are expunged.
seen	seen is a message status flag set by the system when the user opens a message. If the attribute seen is set to and, then the message must be seen **and** other criteria must be met before the rule is fulfilled. If the attribute seen is set to or, then the message only needs to be seen **or** another criteria be met before the rule is fulfilled. (and/or).
sieve	A Sieve test specifying message selection criteria. Example: `Rule17.sieve: header :contains "Subject" "Vigara"`
deleted	deleted is a message status flag set by the system when the user deletes a message. If the attribute deleted is set to and, then the message must be deleted *and* another criteria must be met before the rule is fulfilled. If the attribute deleted is set to or, then the message only needs to be deleted **or** another criteria be met before the rule is fulfilled. (**and/or**)

Localized Mailbox Names in imexpire

The IMAP protocol specifies that mailbox names use modified UTF-7 encoding. Messaging Server supports localized character sets on external interfaces so that mailbox names can be localized. Internally, however, the system converts the localized name to MUTF-7. Thus, a folder that has a localized mailbox name on a client will have a corresponding mailbox file name in MUTF-7. (Note that IMAP error messages will output mailbox names in MUTF-7 and not the localized character set.)

In general, most message store utilities that require mailbox names expect the names in the localized character set, although they may have an option flag that allows a different character set to be used. These utilities include reconstruct, mboxutil, imsbackup, imsrestore, and hashdir. However, imexpire requires that the mailbox name, specified as the attribute folderpattern,be in MUTF-7. Using a localized name will not work.

To obtain the appropriate folderpattern for imexpire it may be necessary to convert a localized mailbox name to the modified UTF-7 equivalent. This can be done using the mboxutil -E command as follows:

```
$ mboxutil -l -p user/han/*

  msgs  Kbytes last msg         partition   quotaroot mailbox

    57     100 2010/04/29 11:18 primary       5242880 user/han/INBOX
     1       1 2010/04/30 12:56 primary             -
user/han/<multibyte_mailbox>

$ mboxutil -l -E MUTF-7 -p user/han/*

    57     100 2010/04/29 11:18 primary       5242880 user/han/INBOX
     1       1 2010/04/30 12:56 primary             -
user/han/&kAFP4W4IMH8wojCkMMYw4A-
```

The first mboxutil shows the localized mailbox name. The second mboxutil shows the mailbox name in MUTF-7. The MUTF-7 mailbox name is identical to the IMAP list command:

```
x list "" *
* LIST (\NoInferiors) "/" INBOX
* LIST (\HasNoChildren) "/" &kAFP4W4IMH8wojCkMMYw4A-
```

To convert the local charset to modified UTF-7 encoding, use the `mboxutil` command with the `-E` option:

```
$ mboxutil -l -E MUTF-7 -P user/han/<multibyte_mailbox>

  msgs  Kbytes last msg         partition   quotaroot mailbox

     1       1 2010/04/30 12:56 primary             -
user/han/&kAFP4W4IMH8wojCkMMYw4A-
```

Note that `mboxutil -E` can be used for any command that requires the use of a MUTF-7 mailbox name including `imexpire`.

Setting imexpire Rules Textually

Message expiration rules are set by specifying expire criteria in a `store.expirerule` file. The `store.expirerule` file contains one expire criteria per line. An expire criteria of the global rule configuration file (*msg-svr-base*`/data/store/store.expirerule`) has the following format:
rule_name`.`*attribute*`:` *value*

An expiration rule for a user or mailbox rule configuration file has this format:
attribute`:` *value*

Example `imexpire` Rules shows a set of global expiration rules in *msg-svr-base* `/config/store.expirerule`.

Rule 1 sets the global expiration policy (that is, policy that applies to all messages), as follows:

- Enable UNIX regular expressions in rules creation.
- Removes messages larger than 100,000 bytes after 3 days.
- Removes messages deleted by the user.
- Removes any message with the strings "Vigara Now!" or "XXX Porn!" in the Subject: header.
- Limits all folders to 1,000 messages. After 1,000 messages, the system removes the oldest messages on a folder to keep the total to 1,000.
- Removes all messages older than 365 days.

Rule 2 sets the message expiration policy for users at the hosted domain `siroe.com`. It limits mailbox sizes to 1 megabyte, removes messages that have been deleted, and removes messages older than 14 days.

Rule 3 sets the message expiration policy for messages in the `inbox` folder of user `f.dostoevski`. It removes messages with a subject line having the expression "On-line Casino."

Example: Example `imexpire` Rules

```
Rule1.regexp: 1
Rule1.folderpattern: user/.*
Rule1.messagesize: 100000
Rule1.messagesizedays: 3
Rule1.deleted: or
Rule1.Subject: Vigara Now!
Rule1.Subject: XXX Porn!
Rule1.messagecount: 1000
Rule1.messagedays: 365
Rule2.regexp: 1
Rule2.folderpattern: user/.*@siroe.com/.*
Rule2.exclusive: 1
Rule2.deleted: or
Rule2.messagedays: 14
Rule2.messagecount: 1000
Rule3.folderpattern: user/f.dostoevski/inbox
Rule3.Subject: *On-line Casino*
```

Setting imexpire Folder Patterns

Folder patterns can be specified using POSIX regular expressions by setting the `imexpire` attribute `regex` to 1. If not specified, IMAP expressions will be used. The format must start with a `user/` followed by a pattern. The following table shows the folder pattern for various folders.

imexpire Folder Patterns Using Regular Expressions

Scope	Folder Pattern (regex=0)	Folder Pattern (regex=1)
Apply rule to all messages in all folders of *userid*.	`user/`*userid*`/*`	`user/`*userid*`/.*`
Apply rule to messages of *userid* in folder `Sent:`.	`user/`*userid*`/Sent`	`user/`*userid*`/Sent`
Apply rule to entire message store.	`user/*`	`user/.*`
Apply rule to any folder called `Trash` anywhere in any user's hierarchy.	`user/*/Trash`	`user/.*/Trash`
Apply rule to folders in hosted domain `siroe.com`	`user/*@siroe.com/*`	`user/.*@siroe.com/.*`
Apply rule to folders in default domain.	Not applicable	`user/[^@]*/.*`

Message Store Message Types Overview

Message Type Overview

This information provides the conceptual background for using message types in the message store. See Managing Message Types in the Message Store for task information.

Topics:

About Message Type

A unified messaging application can receive, send, store, and administer messages of many types, including text messages, voice mail, fax mail, image data, and other data formats. The message store allows you to define up to 63 different message types.

One method of managing messages by type is to organize the messages by their types into individual folders.

With the introduction of the message type feature, you do not have to maintain different message types in individual mailbox folders. Once you configure a message type, the message store can identify it, no matter where it is stored. Thus, you can store heterogeneous message types in the same folder. You also can perform the following tasks:

- Track the usage of message types
- Send notifications grouped by message type
- Set and administer different quotas for different message types, whether they are stored in the same folder or different folders
- Move messages from one folder to another, according to criteria configured uniquely for each message type
- Expire messages according to criteria configured for each message type

Planning the Message-Type Configuration

In a unified messaging application, data of heterogeneous formats are given standard internet message headers so that Messaging Server can store and manage the data. For example, when voice mail is sent to an end-user's phone, a telephone front-end system adds a message header to the incoming voice mail and delivers it to the message store.

To recognize and administer messages of different types, all components of the unified messaging system must use the same message-type definitions and the same header fields to identify the messages.

Before you configure the message store to support message types, you must:

- Plan which message types you intend to use
- Decide on the definition for each message type
- Decide which header field to use

For example, if your application includes phone messages, you can define this message type as “multipart/voice-message” and use the Content-Type header field to identify message types.

You would then configure the telephone front-end system to add the following header information to each phone message to be delivered to the message store:

```
Content-Type: multipart/voice-message
```

Next, you would configure the message store to recognize the `multipart/voice-message` message type, as described in the sections that follow.

Defining and Using Message Types

You define a message type by giving it a unique definition such as `multipart/voice-message`. By default, the message store reads the Content-Type header field to determine the message-type. If you prefer, you can configure a different header field to identify the message types.

The message store reads the Content-Type (or other specified) header field, ignoring case. That is, the message store accepts the header field as valid even if the header's combination of uppercase and lowercase letters differs from the expected combination.

The message store reads only the message-type name in the header field. It ignores additional arguments or parameters.

To define a message type, use the `configutil` utility to set values for the `store.messagetype` parameters. For instructions, see To Configure Message Types.

Configuring a message type allows the message store to identify and manipulate messages of the specified type. It is the first, essential step in administering message types in a unified messaging application.

To take full advantage of the message-type features provided by the message store, you also should perform some or all of the following tasks:

- Configure a JMQ notification plug-in and write Message Queue clients for retrieving notifications that track the status of the message types
- Configure quota roots that apply to each message type
- Write expire rules and set LDAP attribute values to expire and purge messages according to message type

Message Types in IMAP Commands

When you configure the `store.messagetype.`*`x`*`.flagname` parameter for a message type, you create a unique flag that identifies the message type. This flag cannot be modified by end users.

Messaging Server presents the message-type flag as a user flag to IMAP clients. Mapping the message type to a user flag allows mail clients to use simple IMAP commands to manipulate messages by message type.

For example, you can perform the following operations:

- Use the IMAP FETCH FLAGS command to display a message-type flag name as a user-defined flag to the client.
  For a sample use of the IMAP FETCH FLAGS command, see Example 1, shown below.
- Use a message-type flag as a keyword in an IMAP SEARCH command.
  For a sample use of the IMAP SEARCH command, see Example 2, shown below.

The message-type user flag is read only. It cannot be modified by IMAP commands.

The following examples assume you configure the message-type `configutil` parameters with the values shown here:

```
store.messagetype.enable = yes
store.messagetype.1 = text/plain
store.messagetype.1.flagname = text
store.messagetype.1.quotaroot = text
store.messagetype.2 = multipart/voice-message
store.messagetype.2.flagname = voice_message
store.messagetype.2.quotaroot = voice
```

Example 1: IMAP FETCH Session Based on the Message-Type configutil Configurations

The following IMAP session fetches messages for the currently selected mailbox:

```
2 fetch 1:2 (flags rfc822)
* 1 FETCH (FLAGS (\Seen text) RFC822 {164}
Date: Wed, 8 July 2006 03:39:57 -0700 (PDT)
From: bob.smith@siroe.com
To: john.doe@siroe.com
Subject:  Hello
Content-Type: TEXT/plain; charset=us-ascii
* 2 FETCH (FLAGS (\Seen voice_message) RFC822 {164}
Date: Wed, 8 July 2006 04:17:22 -0700 (PDT)
From: sally.lee@siroe.com
To: john.doe@siroe.com
Subject:  Our Meeting
Content-Type: MULTIPART/voice-message; ver=2.0
2 OK COMPLETED
```

In the preceding example, two messages are fetched, one text message and one voice mail.

The message-type flags are displayed in the format configured with the `store.messagetype.*.flagname` parameter.

The Content-Type header fields identify the message types. The message-type names are displayed as they were received in the incoming messages. They use mixed uppercase and lowercase letters and include the message-type arguments such as `charset=us-ascii`.

Example 2: IMAP SEARCH Session Based on the Message-Type configutil Configurations

The following IMAP session searches for voice messages for the currently selected mailbox:

```
3 search keyword voice_message
* SEARCH 2 4 6
3 OK COMPLETED
```

In the preceding example, messages 2, 4, and 6 are voice messages. The keyword used in the search is `voice_message`, the value of the `store.messagetype.2.flagname` parameter.

Message Store Quota (Overview)

Message Store Quota--Overview and Concepts

This information describes quota concepts. See Managing Message Store Quotas for information on how to use quotas in your system.

Message store quotas limit or reduce message store usage. They enable you to set *quotas* for how much disk space or how many messages can be used by a user or domain.

Topics:

Quota Overview

Quotas can be set, in terms of number of messages or number of bytes or both, for specific users or domains. Quotas can also be set for specific folders and message types. For example, you can set different quotas based whether a message is a voice mail or an email. Folder quotas set limits to the size of a user's folder in bytes or messages. For example, a quota can be set on the Trash folder. Messaging Server enables you to set default quotas for domains and users as well as customized quotas.

You can also configure how the system responds to users or domains that are either over quota or approaching the quota. One response is to send users an *over quota notification*. Another response is to halt delivery of messages into the message store when quota is exceeded. This is called *quota enforcement* and usually occurs after a specified *grace period*. A grace period is how long the mailbox can be over the quota before enforcement occurs. If message delivery is halted due to over quota, incoming messages can either remain in the MTA queue until one of the following occurs or be rejected by my the MTA immediately, if `local.store.overquotastatus` is enabled:

- The size or number of the user's messages no longer exceeds the quota, at which time the MTA delivers the messages.
- The undelivered message remains in the MTA queue longer than the specified *grace period*, at which time messages are returned to sender. (See To Set a Grace Period).
- The message has remained in the message queue longer than the maximum message queue time. This is controlled by the `notices` MTA channel keyword (see To Set Notification Message Delivery Intervals).

For example, if your grace period is set for two days, and you exceed quota for one day, new messages continue to be received and held in the message queue, and delivery attempts continue. After the second day, the messages bounce back to the sender.

Disk space becomes available when a user deletes and expunges messages or when the server deletes messages according to expiration policies established (see Message Store Message Expiration).

Exceptions for Telephony Application Servers

To support unified messaging requirements, Messaging Server provides the ability to override quota limitations imposed by the message store. This guarantees the delivery of messages that have been accepted by certain agents, namely telephony application servers (TAS). Messages accepted by a TAS can be routed through a special MTA channel that ensures the message is delivered to the store regardless of quota limits. This is a fairly esoteric usage, but can be useful to telephony applications. For more information about configuring a TAS channel, contact your Oracle messaging representative.

Quota by message type is useful for telephony applications that use unified messaging. For example, if a mix of messages, say text and voice mail, is stored in a user's mailbox, then the administrator can set different quotas for different types of messages. One quota can be set for email and another can be set for voice mail.

Quota Theory of Operations

Customized user and domain quotas are specified by adding quota attributes to LDAP user and domain entries. Quota defaults, notification policy, enforcement, and grace period are specified in `configutil` parameters or by using the `imquotacheck` command.

To determine if a user is over quota, Messaging Server first determines if a quota has been set for the individual user. If no quota has been set, Messaging Server looks at the default quota set for all users. For a user, the quota is for all the cumulative bytes or messages in all of the user's folders. For a domain, the quota is for all the cumulative bytes or messages of all the users in a particular domain. For a message type, the quota is for all the cumulative bytes or messages for that message type. For a folder, the quota is for all the cumulative bytes or messages for user's folder.

You can specify the following quota values for a user's mailbox tree:

- Quota values for specific folders in the user's mailbox.
- Quota values for specific message types such as voice mail or text messages. (A message type quota applies to messages of that type in all folders in the user's mailbox.)
- A default quota value that applies to all folders and message types in the user's mailbox that are not explicitly assigned quotas.

The following guidelines apply when you assign multiple quota values for a user:

- Quotas do not overlap. For example, when there is a quota for a particular message type or folder, messages of that type or messages in that folder are not counted toward the default quota. Each message counts toward one and only one quota.
- The total quota for the whole user mailbox equals the sum of the values of all the quotas specified by default, type, and folder.
- Message type quotas take precedence over folder quotas. For example, suppose one quota is specified for a user's `memos` folder and another quota is specified for voice messages. Now suppose the user stores eight voice messages in the `memos` folder. The eight messages are counted toward the voice mail quota and excluded from the `memos` folder quota.

Changes made to quota attributes, `mailDomainDiskQuota`, `mailDomainMsgQuota`, and `mailDomainStatus` take effect immediately. Changes to quota attributes, `mailQuota`, `mailMsgQuota`, `mailUserStatus`, and `store.quotagraceperiod` only take effect after ims_master automatically restarts.

In order for changes to other `configutil` parameters to take effect, restart Messaging Server. Messaging Server provides a command, `iminitquota` that updates the changes immediately. The `imquotacheck` utility enables you to check message store usage against assigned quotas.

Methods of Notification

If `store.quotanotification` is enabled, when users approach or exceed their quota limit (depending on `store.quotawarn`), the message defined by `store.quotaexceededmsg` notifies them immediately. Otherwise, you must run imquotacheck -n to notify the users. Dynamic user notification by enabling `store.quotanotification` and running `imquotacheck -n` are mutually exclusive. If `store.quotanotification` is enabled, you should not use `imquotacheck -n`. The preferred method is dynamic notification.

Domain level quota enforcement and reporting is done by running `imquotacheck -f`.

Message Store Quota Attributes and Parameters

This section lists the major the message store quota attributes and `configutil` parameters. The intention is to provide you with an overview of the functionality interface. For detailed information on these attributes and parameters, refer to the appropriate reference documentation.

The following table lists the quota attributes. Refer to Messaging Server, Calendar Server, and Contacts Server LDAP Object Classes and Attributes.

Message Store Quota Attributes

Attribute	Description
`mailQuota`	Disk space allowed for the user's mailbox.
`mailMsgQuota`	Maximum number of messages permitted for a user. This is a cumulative count for all folders in the store.
`mailUserStatus`	Status of the mail user. Some of the possible values are `active`, `inactive`, `deleted`, `hold`, and `overquota`.
`mailDomainDiskQuota`	Disk space allowed for the cumulative count of all the mailboxes in a domain.
`mailDomainMsgQuota`	Maximum number of messages permitted for a domain, that is, the total count for all mailboxes in the store.
`mailDomainStatus`	Status of the mail domain. Values and default are the same as `mailUserStatus`.

To have the preceding attributes take effect, run `iminitquota` to initialize user's quota settings and make quota and usage up-to-date. The changes would take effect without running this command, but not immediately, as information is stored in caches and it takes a little time before the changes take effect.

For the changes made to `configutil` parameters related to quota, changes on `store.quotagraceperiod` will take effect automatically after ims_master's automatically restart. Changes on other `configutil` parameters will only take effect after restarting Messaging Server.

The following table lists the quota parameters. Refer to Messaging Server Configuration for the latest and most detailed information.

Message Store `configutil` parameters

Parameter	Description
store.quotaenforcement	Enable quota enforcement When off, the quota database is still updated, but messages are always delivered. Default: On
store.quotanotification	Enable quota notification. Default: OFF
store.defaultmailboxquota	Store default quota by number of bytes. Format is num[K|M|G]. Maximum is 4294967292K. Default: -1 (unlimited)
store.defaultmessagequota	Store default quota by number of messages. Numeric. Default: -1 (unlimited)
store.quotaexceededmsg	Message to be sent to user when quota exceeds store.quotawarn. If none, notification is not sent. Default: None. The message must contain a header (with at least a subject line), followed by $$, then the message body. The $ represents a new line. There is support for the following variables: [ID] - userid, [DISKUSAGE] - disk usage, [NUMMSG] - number of messages, [PERCENT] - store.quotawarn percentage, [QUOTA] - mailquota attribute, [MSGQUOTA] - mailmsgquota attribute.
store.quotaexceededmsginterval	Interval, in days, for sending overquota notification. Default: 7
store.quotagraceperiod	Time, in hours, a mailbox has been overquota before messages to the mailbox will bounce back to the sender. Number of hours. Default: 120
store.quotawarn	Quota warning threshold. Percentage of quota exceeded before clients are sent an over quota warning. Default: 90
local.store.quotaoverdraft	Used to provide compatibility with systems that migrated from the Netscape Messaging Server. When ON, allow delivery of one message that puts disk usage over quota. After the user is over quota, messages are deferred or bounced, the quota warning message is sent, and the quota grace period timer starts. (The default is that the quota warning messages are sent when the message store reaches the threshold.) Default: Off, but is treated as on if local.store.overquotastatus is set, otherwise the user can never go over quota and the overquotastatus is never used.
local.store.overquotastatus	Enable quota enforcement before messages are enqueued in the MTA. This prevents the MTA queues from filling up. When set, and a user is not yet over quota, but an incoming message pushes the user over quota, then the message is delivered, but the mailuserstatus LDAP attribute is set to overquota so no more messages are accepted by the MTA. Default: off

To have the preceding configutil parameters take effect, restart Messaging Server.

The imquotacheck utility enables you to check message store usage against assigned quotas.

Also see iminitquota to update the information in the quota database if a user's quota-related LDAP

attributes (or the system defaults) have been changed recently and the changes have not yet been propagated automatically to the quota database.

Message Store Shared Folders Overview

Shared Folders Overview

See Managing Shared Folders for shared folder tasks.

A *shared folder* is like any other mail folder except that users other than its owner can read, delete, or add messages to it, depending on the accress rights they are granted. Messages can be added to shared folders by normal drag and drop, by Sieve filters, or by sending messages directly using the form: *uid+folder@domain*.

The example below shows the address for sending email to a *private shared folder* owned by `carol.fanning@siroe.com` called `crafts_club`:

```
carol.fanning+crafts_club@siroe.com
```

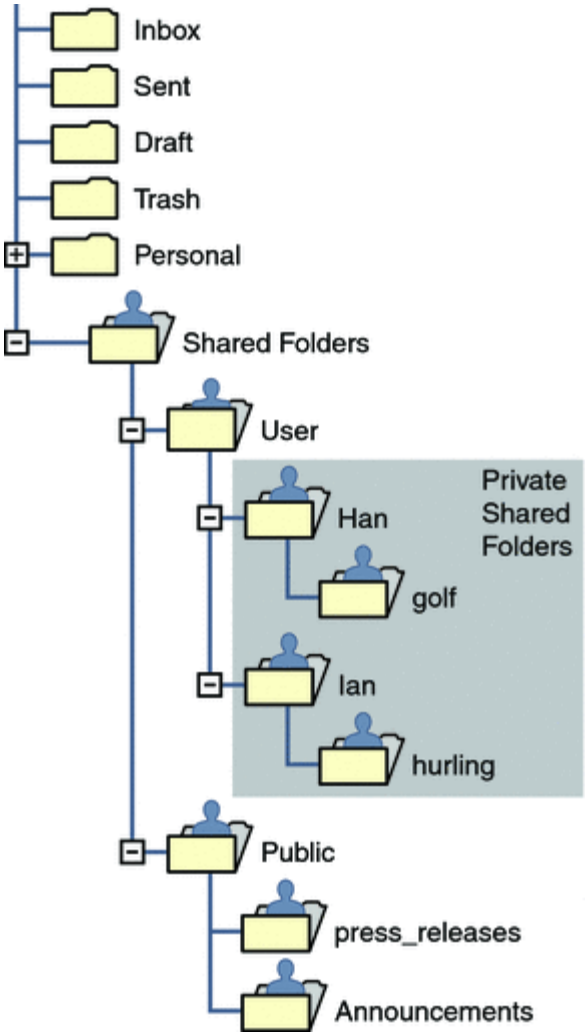
This example shows the address for sending email to a *public shared folder* called `tennis`:

```
public+tennis@siroe.com
```

Shared folders are useful for starting, sharing, and archiving an ongoing email conversation on a particular topic. For example, a group of software developers can create a shared folder for discussing development of a particular project called `mosaic_voices`. When a message is sent or dropped into the folder `mosaic_voices`, anyone who has permissions to access the shared folder (permissions can granted to individuals or groups) can open this mailbox and read the message.

Shared folders are displayed in user's mailbox tree under a folder called `Shared Folders`. An example is shown below.

Example of Shared Mail Folder List as Seen from a Mail Client



There are two kinds of shared folders:

- **Private Shared Folder** – A shared folder created and owned by a specific user with access rights granted to other users or groups. The owner can grant access rights using Communications Express or other mail clients that support shared folder creation. The mail administrator can also grant access rights using the `readership` command. Private shared folders appear in the `Shared Folders/User` mail folder directory.
- **Public Shared Folder** – A shared folder created by the mail administrator and not owned by a specific user. The mail administrator can grant access rights using the `readership` command. Public shared folders appear in the `Shared Folders/Public` mail folder directory.

For example, you might want a folder, such as `public+software_dev@siroe.com` for posting information about a special interest group inside the company. Interested employees would be granted access to this public folder.

Messaging Server allows folders to be shared among users of different backend message stores. See To Set Up Distributed Shared Folders for details.

See Managing Shared Folders for examples of creating shared folders and granting access rights.

Message Store Valid UIDs and Folder Names

Valid Message Store UIDs and Folder Names

This information describes valid constructions for message store UIDs and folder names. Note that folder and mailbox are used synonymously.

Topics:

Message Store User ID

The message store user ID is a mail user's unique identifier in the message store. In the default domain, this is the same as the user's LDAP `uid` attribute. In hosted domains, this is *uid@domain* where *uid* is the `uid` LDAP attribute and *domain* is the canonical domain name.

Message Store Mailbox Name for Commands

Some message store commands require that you specify a mailbox name. The required form of the name is `user/`*`userid`*`/`*`mailbox`* where *userid* is the message store user ID (see Message Store User ID) and *mailbox* is a user's mailbox. Specifying INBOX sometimes implies all the user's mailboxes in the message store. For example, the following command removes the INBOX and all the folders of user `joe`.

```
mboxutil -d user/joe/INBOX
```

Note that in the context of message stores, folders and mailboxes are synonymous.

Valid UIDs

Valid and invalid `UID` characters are controlled separately by the MTA and message store mechanisms. That means `UID` character limitations are specified by the union of MTA and message store limitations. The following characters and strings are invalid as `UIDs` in the message store:

- % ? * & / : \
- ASCII values less than 20 or greater than 7E hexidecimal (see man ascii)
- A leading '-' is prohibited because it is reserved for negative rights
- A leading 'group=' is prohibited because it is reserved for group IDs
- The following UIDs are reserved: 'anonymous' 'anybody' 'anyone' and 'anyone@domain'
- The maximum supported length for a UID is 127 bytes

The following characters are invalid in `UIDs` in the MTA:

```
<space> $ ~ = # * + % ! @ , { } ( ) / \ < > ; : " ` [ ] & ?
```

The list of characters forbidden by the MTA can be modified by setting the `option.dat` parameter `LDAP_UID_INVALID_CHARS` with a string of the forbidden characters using decimal ASCII values, however, you are strongly advised not to change the default constraint. The default setting is as follows and reflect the characters listed above:

```
LDAP_UID_INVALID_CHARS=32,33,34,35,36,37,38,40,41,42,43,44,47,58,59,60,61,6
```

Valid mail folder names. The following characters are invalid as folder names:

% * ? and ASCII values less than 20 or greater than 7E hexidecimal (see `man ascii`).

In addition, folder names must be valid MUTF-7 sequences.

Migrating Mailboxes to a New System

Migrating Mailboxes to a New System

Messaging Server provides several ways to move or relocate mailboxes from one Messaging Server host (mailhost) to another.

Topics:

For More Information

- Reducing Message Store Size Due to Duplicate Storage
- moveuser
- rehostuser

Tools Summary

The following tools are available for relocating users from one mailhost to another:

- **`rehostuser`, Available starting in Messaging Server 7:** Enables you to move a Messaging Server user's mail store from one mailhost to another. The `rehostuser` utility also disconnects any active session, locks the store to ensure atomicity of the move from the user's perspective (no loss of data, flag change, and so on), changes the user's LDAP entry, flushes LDAP caches as necessary, and causes any queued mail to be rerouted to the new store. See `rehostuser` for more information.
- **`MoveUser`:** Moves a user's account from one Messaging Server host to another. When user accounts are moved from one Messaging Server host to another, it is also necessary to move the user's mailboxes and the messages they contain from one host to the other. See `MoveUser` for more information.
- **`imsbackup | imsrestore`:** Manually backs up the account from one host and restores on another. Can be combined with with `ssh` and "piped" across the network. You can also back up to a file and then access that file from the other host through NFS, or move the file in between.
- **`imsimport`:** Imports messages from a UNIX `/var/mail` format file into the Messaging Server message store.
- **Other suggested options:** See http://lists.balius.com/pipermail/info-ims-archive/2006-July/012420.html and http://forums.sun.com/thread.jspa?threadID=5103600&tstart=2086 for ideas using `rsync` and other UNIX tools.

Notes:

- If both the source and destination mailhosts are installed with at least Messaging Server 7, use `rehostuser`. The `rehostuser` utility solves issues present in the other solutions, such as preventing users from accessing mail while it is being moved, making sure mail is held in the MTA during the move and redirected to the new message store, and so on.
- You can use `MoveUser` with non-Oracle IMAP servers as well as all versions of Messaging

Server. If you are moving users from a non-Oracle server and it has IMAP capability, `MoveUser` is a good choice. The `MoveUser` utility also modifies the user's `mailHost` attribute on completion.

- You can manually run `imsbackup` on one host and `imsrestore` on the other host. You can combine multiple users in one step. You can combine `imsbackup` with `ssh` or NFS, or you could use `gzip` and transfer the backup file from one host to the other by using `ftp`. This method is actually used within the `rehostuser` utility. However, `rehostuser` shields you from having to set the users' status, disconnect them, and so on. If you use `imsbackup` and `imsrestore` manually, you also need to deal with those details manually. But if you have a large amount of data to move and can afford for the MTA and IMAP user access to be down while it is being moved, this method might be more efficient.
- Similar to `imsrestore` and `imsbackup`, you can use `imsimport` to import UNIX `/var/mail` format files into the message store. If you are moving from a non-Oracle server and it does not have IMAP access capability, perhaps you can get it to export the folders in UNIX `/var/mail` format and then import them this way.

Migrating Mailboxes From an x86 Host to a SPARC Host

The message store data formats are architecture dependent. You cannot transfer any data components in the Message Store Components matrix from one architecture to another directly. To migrate the message store from an x86 host to a SPARC host (or from a SPARC host to an x86 host), use `imsbackup` and `imsrestore`, or `rehostuser`.

Moving Mailboxes to Another Messaging Server While Online

You can migrate the message store from an older version of Messaging Server to a newer version, or move mailboxes from one Messaging Server message store to another, while remaining online. This procedure works for iPlanet Messaging Server 5.0 and later. You cannot move messages from prioor versions of Messaging Server or a non-Oracle Communications Suite message store. Moving mailboxes while online have the following advantages and disadvantages.

Advantages

- You can move the mailboxes from the old source system to the new destination system without user involvement.
- This process is faster than any of the other processes.
- Re-linking is not required if you are moving an entire partition.
- Both Messaging Server systems remain active and online.
- You can migrate all the mailboxes on a messages store or a subset of those messages. This procedure allows for incremental migrations.

Disadvantages

- This method does not work with non-Oracle Communications Suite messaging servers.
- The users being migrated do not have access to their mailboxes until the migration of their own mailbox is complete.
- This method can be complex and time consuming.

Incremental Mailbox Migration While On-line

Incremental migration provides numerous advantages for safely and effectively moving your message store to a different system or upgrading to a new system, incremental migration allows you to build a new back-end message store system alongside the old back-end message store. You can then test the new system, migrate a few friendly users, then test the new system again. Once you are comfortable with the new system and configuration, and you are comfortable with the migration procedure, you can start migrating real commercial users. These users can be split into discrete backup groups so that during migration, only members of this group are offline, and only for a short time.

Another advantage of on-line incremental migration is that you do not have to plan for a system-wide back out in case your upgrade fails. A back out is a procedure for reverting changes you have made to a system to return the system to the original working state. When doing a migration, you have to plan for failure, which means that for every step in the migration requires a plan to return your system back to its previous operational state.

The problem with offline migrations is that you can't be sure your migration is successful until you've completed all the migration steps and switched the service back on. If the system doesn't work and cannot be quickly fixed, you'll need a back out procedure for all the steps performed. This can be stressful and take some time, during which your users will remain offline.

With an on-line incremental migration you perform the following basic steps:

1. Build the new system alongside the old one so that both can operate independently.
2. Configure the old system for coexistence with the new.
3. Migrate a group of "friendly" users and test the new system and its coexistence with the old system.
4. Divide the users on the old system into groups and migrate group by group to the new one as desired.
5. Disassemble the old system.

Because both systems will coexist, you have time to test and get comfortable with the new system before migrating to it. If you do have to perform a back-out procedure, which should be very unlikely, you only have to plan for steps 2 and 4. Step 2 is easy to revert because you do not ever touch user's data. In step 4, the backout is to revert the user's state to active and their mailhost attribute back to the old host. No system-wide back out is required.

Online Migration Overview

Migrating mailboxes while remaining online is a straightforward process. Complications arise when you try to ensure that messages in transit to the mailbox (sitting in an MTA channel queue waiting for delivery) are not lost in the migration process. One solution is to hold messages sent during the migration process in a *held* state and wait for the messages in the various channel queues to be delivered. However, messages can get stuck in queues because of system problems or because a particular user is over quota. In this case, you must address this situation before migrating the mailboxes.

You can take various measures to reduce the likelihood of lost messages and to verify that messages are not stuck in a channel queue, but at a cost of increased complexity of the procedure.

The order and necessity of steps in the procedure vary depending upon your deployment and whether every message addressed to every mailbox must not be lost. This section describes the theory and concepts behind the steps. It is incumbent on you to understand each step and decide which to take and in which order, given your specific deployment. Following is an overview of the process of moving mailboxes. This process might vary depending upon your deployment.

1. Block user access to the mailboxes being moved.
2. Temporarily hold messages addressed to the mailbox being moved.
3. Verify that messages are not stuck in the channel queues.
4. Change the user's mailhost attribute to the new mailbox location.
5. Move the mailboxes to the new location.
6. Release held mail to be delivered to the new mailbox and enable incoming messages to be delivered to the migrated mailboxes.
7. Examine the old message store to see if any messages were delivered after the migration.
8. Unblock user access to mailbox.

To Migrate User Mailboxes from One Messaging Server to Another While Online

The requirements for this type of migration are as follows:

- `stored` should be running on both the source (old) and destination (new) messaging servers.
- The source system and destination system must be able to route messages to each other if both systems will operate in co-existence. This is needed, for instance, so that delivery status notification messages can be generated on the destination system and get delivered to the source system.

Note
Some steps apply only if you are upgrading the messaging server from an earlier version to a later version. These steps might not apply if you are only migrating mailboxes from one message store to another. The steps that apply to migrating entire systems are noted.

1. On the source system, split your user entries to be moved into equal backup groups by using the `backup-groups.conf` file.
   This step is in preparation for the mailbox migration, Step 8, that occurs later in this procedure. See To Create Backup Groups for detailed instructions.
   You can also place the user names into files and use the -u option in the `imsbackup` command.
2. Notify users to be moved that they will not have access to their mailboxes until the move is completed.
   Ensure that users to be moved are logged out of their mail systems before the data move occurs. (See Monitoring User Access to the Message Store.)
3. Set the authentication cache timeout to 0 on the back-end message store and MMP systems, and `ALIAS_ENTRY_CACHE_TIMEOUT` option to 0 on the MTAs.
   a. On the back-end message stores containing the mailboxes to be moved, set the authentication cache timeout to 0.

      ```
      configutil -o service.authcachettl -v 0
      ```

      This step and Step 7 (changing `mailUserStatus` to `hold`) immediately prevents users from accessing their mailboxes during migration.

   b. On all MMPs, set the LDAP and authentication cache timeout to 0.
      In `ImapProxyAService.cfg` and `PopProxyAService.cfg`, set both `LdapCacheTTL` and `AuthCacheTTL` to 0.
   c. On any Messaging Server host that contains an MTA that inserts messages into mailboxes that are to be migrated, set the `ALIAS_ENTRY_CACHE_TIMEOUT` option to 0.
      Messaging Servers hosting an MTA that inserts messages into the migrating mailboxes will typically be the back-end message store. However, if the system is using LMTP, then that system will be the inbound MTAs. Check your configuration to make sure.
      Resetting `ALIAS_ENTRY_CACHE_TIMEOUT` in *msg-svr-base*`/config/option.dat` forces the MTA to bypass the cache and look directly at the LDAP entry so that intermediate channel queues (for example, the `conversion` or `reprocess` channels) see the new `mailUserStatus` (`hold`) of the users being moved rather then the out-of-date cached information. `ALIAS_ENTRY_CACHE_TIMEOUT` is in `option.dat`.
   d. Restart the systems on which the caches were reset.
      You must restart the system for these changes to take place. See Starting and Stopping Services for instructions.
4. Ensure that both your source Messaging Server and destination Messaging Server are up and running.
   The source Messaging Server must be able to route incoming messages to the new destination server.
5. Change the LDAP attribute `mailUserStatus` on all user entries whose mailboxes will be moved from `active` to `hold`.
   Changing the attribute holds incoming messages in the `hold` queue and prevents access to the mailboxes over IMAP, POP, and HTTP. Typically, users will be moved in groups of users. If you are moving all the mailboxes of a single domain, you can use the `mailDomainStatus` attribute.

For more information on `mailUserStatus`, see *Unified Communications Suite Schema Reference*.

6. Make sure that messages addressed to mailboxes being migrated are not stuck in the `ims-ms` or `tcp_lmtp*` channel queues (if LMTP has been deployed).
   Use the following commands to see if messages exist in the channel queue directory tree and in the *held* state (to see `.HELD` files) addressed to a user to be migrated:

   ```
   imsimta qm directory -to=<user_address_to_be_migrated> -directory_tree
   imsimta qm directory -to=<user_address_to_be_migrated> -held
   -directory_tree
   ```

   If there are messages in the queue, run these same commands later to see if the MTA has dequeued them. If there are messages that are not being dequeued, then you must address this problem before migrating. This should be a rare occurrence, but possible causes are recipient mailboxes being over quota, mailboxes being locked perhaps because users are logged in and moving messages, the LMTP backend server is not responding, network or name server problems, and so on).

7. Change the LDAP attribute `mailHost` in the user entries to be moved as well as in any mail group entries.
   Use the `ldapmodify` command to change the entries to the new mail server. Use the `ldapmodify` that comes with Messaging or Directory Server. Do NOT use the Oracle Solaris `ldapmodify` command.
   - You only need to change the `mailHost` attribute in the mail group entry if the old mail host is being shut down. You can either change this attribute to the new mail host name or just eliminate the attribute altogether. It is optional for mail groups to have a `mailHost.` Having a `mailHost` means that only that host can do the group expansion; omitting a `mailHost` (which is the more common case) means all MTAs can do the group expansion. Note that mail group entries do not have mailboxes to be migrated and typically do not even have the `mailhost` attribute.
     For more information on `mailhost`, see *Unified Communications Suite Schema Reference* .
8. Move the mailbox data from the source Messaging Server message store to the destination Messaging Server message store and record the time when started.
   Back up the mailboxes with the `imsbackup` utility and restore them to the new Messaging Server with the `imsrestore` utility. For example, to migrate mailboxes from a Messaging Server 5.2 system called `oldmail.siroe.com` to `newmail.siroe.com`, run the following command on `oldmail.siroe.com`:

   ```
   <server-root>/bin/msg/store/bin/imsbackup -f- instance/group | rsh
   newmail.siroe.com /opt/SUNWmsgsr/lib/msg/imsrestore.sh -f- -c y -v 1
   ```

   You can run multiple concurrent `imsbackup` and `imsrestore` sessions (one per group) to maximize the transfer rate into the new message store. See also Backing Up and Restoring the Message Store.

   Note
   When `imsrestore` or any processing intensive operation takes significantly more system resources than normal, and continues doing so longer than the `msprobe` interval, there may be a temporary backlog of DB transaction log files to be cleared. If there are more files than specified in `local.store.maxlog`, then `msprobe` may erroneously restart all the processes during a restore. To prevent this from happening, disable `msprobe` during the `imsrestore`.

> **Note**
> Record the timestamp of when `imsbackup` is run for later delivery validation.

9. (*Conditional Step for System Upgrades*) If your mailbox migration is part of the process of upgrading from an earlier version of Messaging Server to the current version, set this current version of Messaging Server to be the new default Messaging Server for the system.
   Change the DNS A record of `oldmail.siroe.com` to point to `newmail.siroe.com` (the server responsible for domain(s) previously hosted on `oldmail.siroe.com`).
10. Enable user access to the new message store.
    Set the LDAP attribute `mailUserStatus` or `mailDomainStatus`, if applicable, to whatever value it had been before it was changed to `hold` (for example,`active`).
11. Release the messages in the *held* state on all source Messaging Servers.
    Any system that may be holding incoming messages needs to run the following command to release all the user messages:

    ```
    imsimta qm release -channel=hold -scope
    ```

    where *scope* can be `all`, which releases all messages; `user`, which is the user ID; or `domain` which is the domain where the user resides.

12. Reset the authentication cache timeout and the `ALIAS_ENTRY_CACHE_TIMEOUT` option to the default or desired values and restart the system.
    At this point, you've migrated all the user mailboxes that need to be migrated. Before proceeding, make sure that no new entries in LDAP have been created with the old system as the `mailhost`, and if some have, migrate them. Also, make sure that no such entries can be created by modifying the provisioning systems.
    You also want to change the `preferredmailhost` attribute to the name of the new mail host.
    For back-end messages stores, set authentication cache timeout as follows:

    ```
    configutil -o service.authcachettl -v 900
    ```

    For the MMPs, in `ImapProxyAService.cfg` and `PopProxyAService.cfg` set the `LdapCacheTTL` and `AuthCacheTTL` options to 900.
    For MTAs, set the `ALIAS_ENTRY_CACHE_TIMEOUT` option to 600.
    `ALIAS_ENTRY_CACHE_TIMEOUT` is in `option.dat`.
    You must restart the system for these changes to take place. See Starting and Stopping Services for instructions.

13. Ensure that the user clients are pointing to the new mail server.
    After the upgrade finishes, have the users point to the new server through their mail client program (in this example, users would point to `newmail.siroe.com` from `oldmail.siroe.com`).
    An alternative is to use a messaging multiplexor (MMP) which obviates the need to have users point their clients directly to the new mail server. The MMP gets that information from the `mailHost` attribute which is stored in the LDAP user entries and automatically redirects the client to the new server.
14. After everything works, verify that no messages were delivered to the old message store after the migration.
    Go to the old message store and run `mboxutil -l` to list the mailboxes. Check the last message delivery timestamp. If a message was delivered after the migration timestamp (the date stamp when you ran the `imsbackup` command), then migrate those messages with a backup and restore command. Because of the preparatory steps provided, it would be exceedingly rare to see a message delivered after migration.

Theoretically, a message could be stuck in a queue for the number of days or hours specified by the `notices` channel keywords (see To Set Notification Message Delivery Intervals).

15. Remove duplicate messages on the new message store, run the `relinker` command.
    This command might free disk space on the new message store. See Reducing Message Store Size Due to Duplicate Storage.
16. Remove the old messages from the store you migrated from and delete users from the database on the old store.
    Run the `mboxutil -d` command. (See the mboxutil page).

To Move Mailboxes Using an IMAP client

This procedure can be used anytime messages need to be migrated from one messaging server to a different messaging server. Consider the advantages and disadvantages before moving mailboxes using this method.

Advantages

- This method can be used to migrate from a non-Oracle Communications host to the Messaging Server host. It can also be used to move mailboxes from one physical server to a different physical server.
- After you set up the new mail server or message store, responsibility for moving mailboxes to the new system is left to users.
- The process for moving mailboxes is relatively simple.
- User access to mailboxes does not have to be disabled.

Disadvantages

- Requires that both the old and new systems be simultaneously running and accessible to users.
- Cumulatively, this method takes longer to move mailboxes than other methods.
- Responsibility for moving mailboxes to the new system is left to users.
- The size of the new message store will be significantly larger than the old message store until the re-linking operation is performed.

1. Install and configure the new Messaging Server.
2. Set `local.store.relinker.enabled` to `yes`.
   This will reduce the message store size on the new system caused by duplicate storage of identical messages. See Reducing Message Store Size Due to Duplicate Storage for more information.
3. Provision users on the new Messaging Server.
   You can use Delegated Administrator to do this. As soon as users are provisioned on the new system, newly arriving mail will be delivered to the new INBOX.
4. Have users configure their mail client to view both new and old Messaging Server mailboxes.
   This may involve setting up a new email account on the client. See mail client documentation for details.
5. Instruct users to drag folders from their old Messaging Server to their new Messaging Server.
6. Verify with users that all mailboxes are migrated to the new system, then shut down the user account on the old system.

To Move Mailboxes By Using the `MoveUser` Command

This procedure can be used anytime you need to migrate messages from one messaging server to a different messaging server. It is useful for migrating IMAP mailboxes from a non-Oracle Communications Suite host to the Messaging Server host. Consider the advantages and disadvantages before moving mailboxes using this method.

Advantages

- You have complete responsibility for moving mailboxes from the old system to the new system.

Users do not have to do anything.
- Works with any IMAP servers.

Disadvantages

- Requires that both the old and new systems be simultaneously running and accessible to users.
- This method takes longer to move mailboxes than the other non-IMAP methods.
- Users access to mailboxes must be disabled while mailboxes are being moved.
- The size of the new message store will be significantly larger than the old message store until the re-linking operation is performed.

1. Install and configure the new Messaging Server.
2. Set `local.store.relinker.enabled` to `yes`.
   This will reduce the message store size on the new system caused by duplicate storage of identical messages. See Reducing Message Store Size Due to Duplicate Storage of Identical Messages for more information.
3. Halt incoming mail to the messaging servers.
   Set the user attribute `mailUserStatus` to `hold`.
4. Provision users on the new Messaging Server if needed.
   If you are migrating from a previous version of messaging server, you can use the same LDAP directory and server. `MoveUser` changes the `mailhost` attribute in each user entry.
5. Run the `MoveUser` command.
   To move all users from `host1` to `host2`, based on account information in the Directory Server `siroe.com`:

   ```
   MoveUser -l \
   "ldap://siroe.com:389/o=siroe.com???(mailhost=host1.domain.com)" \
   -D "cn=Directory Manager" -w password -s host1 -x admin \
   -p password -d host2 -a admin -v password
   ```

   See moveuser for details.

6. Enable user access to the new messaging store.
   Set the `mailUserStatus` LDAP attribute to `active`.
7. Shut down the old system.

To Move Mailboxes By Using the `imsimport` Command

This procedure is specifically used to move mailboxes from UNIX `/var/mail` format folders into a Sun Java System Messaging Server message store. However, if the messaging server from which you are migrating can convert the IMAP message stores to UNIX `/var/mail` format, then you can use the `imsimport` command to migrate messages to the Sun Java System Messaging Server. Consider the advantages and disadvantages before moving mailboxes using this method.

Advantages

- You have complete responsibility for moving mailboxes from the old system to the new system. Users do not have to do anything.

Disadvantages

- This method takes longer to move mailboxes than the other non-IMAP methods.
- Users access to mailboxes must be disabled while mailboxes are being moved.
- The size of the new message store will be significantly larger than the old message store until the re-linking operation is performed.

1. Install and configure the new Messaging Server.
2. Set `local.store.relinker.enabled` to `yes`.
   This will reduce the message store size on the new system caused by duplicate storage of identical messages. See Reducing Message Store Size Due to Duplicate Storage for more information.
3. Provision users on the new Messaging Server if needed.
   You can use Delegated Administrator to do this. Do not switch over to the new system yet.
4. Disable user access to both the new and old messaging store.
   Set the `mailUserStatus` LDAP attribute to `hold`. User's mail is sent to the hold queue and access to the mailbox over IMAP, POP, and HTTP is disallowed. MTA and Message Access Servers on the store server must comply with this requirement. This setting overrides any other `mailDeliveryOption` settings.
5. If the mail store from the existing mail server is not already in the `/var/mail` format, convert the mail store to `/var/mail` files.
   Refer to the third-party mail server documentation.
6. Run the `imsimport` command.
   For example:

   ```
   imsimport -s /var/mail/joe -d INBOX -u joe
   ```

   See the imsimport for details.

7. Enable user access to the message store.
   Set the `mailUserStatus` LDAP attribute to `active`.
8. Enable user access to the new messaging store.
9. Shut down the old system.

Monitoring Disk Space

Monitoring Disk Space

This information describes how to monitor disk space and partition usage, and under what circumstances the system should send a warning.

Topics:

Disk Space Usage Overview

Inadequate disk space is one of the most common causes of the mail server problems and failure. Without space to write to the MTA queues or to the message store, the mail server will fail. In addition, unless log files are monitored and cleaned up, they can grow uncontrollably filling up all disk space.

Message store partitions grow as new messages are delivered to the mailboxes; for example, if message store quotas are not enforced, the message store can outgrow the disk space available for a partition. Another cause of running out of disk space are the MTA message queues growing too large. A third area of concern is if a problem occurs with the log file monitoring facilities and the log files growing uncontrollably. (Note that there are a number of log files such as LDAP, MTA, and Message Access, and that each of these log files can be stored on different disks.)

Symptoms of Disk Space Problems

Different symptoms can occur depending on which disk or partition is running out of space. MTA queues can overflow and reject SMTP connections, messages might remain in the `ims-master` queue and not be not delivered to the message store, and log files can overflow.

If a message store partition fills up, message access daemons can fail, and message store data can be corrupted. Message store maintenance utilities such as `imexpire` and `reconstruct` can repair the damage and reduce disk usage. However, these utilities require additional disk space, and repairing a partition that has filled an entire disk can potentially cause down time.

To Monitor Disk Space

Depending upon the system configuration you may need to monitor various disks and partitions. For example, MTA queues may reside on one disk/partition, message stores may reside on another, and log files may reside on yet another. Each of these spaces will require monitoring and the methods to monitor these spaces may differ.

Messaging Server provides specific methods for monitoring message store disk usage and preventing partitions from filling up all available disk space.

You can take the following steps to monitor the message store's use of disk space:

- Set parameters to monitor message store disk usage
- Lock message store partitions when a disk-usage threshold is reached

To Monitor the Message Store

It is recommended that message store disk usage not exceed 75% capacity. You can monitor message store disk usage by configuring the following alarm attributes using the `configutil` utility:

- `alarm.diskavail.msgalarmstatinterval`
- `alarm.diskavail.msgalarmthreshold`
- `alarm.diskavail.msgalarmwarninginterval`
- `alarm.diskavail.msgalarmdescription`

By setting these parameters, you can specify how often the system should monitor disk space and under what circumstances the system should send a warning. For example, if you want the system to monitor disk space every 600 seconds, specify the following command:

```
configutil -o alarm.diskavail.msgalarmstatinterval -v 600
```

If you want to receive a warning whenever available disk space falls below 20%, specify the following command:

```
configutil -o alarm.diskavail.msgalarmthreshold -v 20
```

Refer to http://msg.wikidoc.info/index.php/Configutil_Reference for more information on these parameters.

Monitoring Message Store Partitions

You can halt messages from being delivered to a message store partition when the partition fills more than a specified percentage of available disk space. This is done by setting two `configutil` parameters to enable the feature and specify the disk-usage threshold.

With this feature, the message store daemon monitors the partition's disk usage. As disk usage increases, the store daemon dynamically checks the partition more frequently (ranging from once every 100 minutes to once a minute).

If disk usage goes higher than the specified threshold, the store daemon:

- Locks the partition. Incoming messages are held in the MTA message queue, but not delivered to the mailboxes in the message store partition.
- Logs a message to the default log file.
- Sends an email notification to the postmaster. (You can change the recipient of the email by setting the `configutil` parameter `alarm.msgalarmnoticercpt`.)

When disk usage falls below the threshold, the partition is unlocked, and messages are again delivered to the store.

The `configutil` parameters are as follows:

- `local.store.checkdiskusage` enables the partition-monitoring feature.
  Allowable values: `yes`, `no`
  Default value: `yes`
- `local.store.diskusagethreshold` specifies the disk-usage threshold. The value of `local.store.diskusagethreshold` is a percentage from 1 to 99.
  Default value: 99

You should set the disk-usage threshold to a percentage low enough to give you time to repartition or assign more disk space to the local message store.

For example, suppose a partition fills up disk space at a rate of 2 percent per hour, and it takes an hour to allocate additional disk space for the local message store. In this case, you should set the disk-usage threshold to a value lower than 98 percent.

Refer to http://msg.wikidoc.info/index.php/Configutil_Reference for more information on these parameters.

Monitoring the MTA Queues and Logging Space

You will need to monitor MTA queue disk and logging space disk usage.

For information on managing logging space, see Managing Logging. For example, to learn how to monitor the `mail.log` file, see Managing MTA Message and Connection Logs.

Monitoring the Message Store

Monitoring the Message Store

This information describes message store monitoring tasks. See Message Store Management for conceptual information.

Topics:

- General Message Store Monitoring Procedures
- Monitoring imapd, popd and httpd
- Monitoring `stored`
- Monitoring the State of Message Store Database Locks
- To Monitor Mailbox Quotas and Usage
- To Monitor Message Store Database Statistics with imcheck
- Gathering Message Store Counter Statistics by Using counterutil

For More Information

- Monitoring Disk Space
- Monitoring User Access to the Message Store
- Message Store Logging

General Message Store Monitoring Procedures

This section outlines standard monitoring procedures for the message store. These procedures are helpful for general message store checks, testing, and standard maintenance.

Topics in this section:

- Checking Hardware Space
- Checking Log Files
- Checking User IMAP/POP/Webmail Session by Using Telemetry
- Checking stored Processes
- Checking Database Log Files
- Checking User Folders
- Checking for Core Files

Checking Hardware Space

A message store should have enough additional disk space and hardware resources. When the message store is near the maximum limit of disk space and hardware space, problems might occur within the message store.

Inadequate disk space is one of the most common causes of the mail server problems and failure. Without space to write to the message store, the mail server will fail. In addition, when the available disk space goes below a certain threshold, there will be problems related to message delivery, logging, and so forth. Disk space can be rapidly depleted when the clean up function of the `stored` process fails and deleted messages are not expunged from the message store.

For information on monitoring disk space, see Monitoring Disk Space.

Checking Log Files

Check the log files to make sure the message store processes are running as configured. Messaging Server creates a separate set of log files for each of the major protocols, or services, it supports: SMTP, IMAP, POP, and HTTP. You can look at the log files in the *msg-svr-base*`/log/` directoy. You should monitor the log files on a routine basis.

Be aware that logging can impact server performance. The more verbose the logging you specify, the more disk space your log files will occupy for a given amount of time. You should define effective but realistic log rotation, expiration, and backup policies for your server. For information about defining logging policies for your server, see Message Store Logging.

Checking User IMAP/POP/Webmail Session by Using Telemetry

Messaging Server provides a feature called telemetry that can capture a user's entire IMAP, POP or HTTP session into a file. This feature is useful for debugging client problems. For example, if users complain that their message access client is not working as expected, this feature can be used to trace the interaction between the access client and Messaging Server.

To capture a POP session, create the following directory:
msg-svr-base`/data/telemetry/`*pop_or_imap_or_http*/*userid*

To capture a POP session, create the following directory:

msg-svr-base`/data/telemetry/pop/`*userid*

To capture an IMAP session, create the following directory:

msg-svr-base`/data/telemetry/imap/`*userid*

To capture a Webmail session, create the following directory:

msg-svr-base`/data/telemetry/http/`*userid*

Note:*userid* is "uid" for default domain and "uid@domain" for hosted domains.

Note that the directory must be owned or writable by the messaging server userid.

Messaging Server will create one file per session in that directory. Example output is shown below.

```
LOGIN redb 2003/11/26 13:03:21
>0.017>1 OK User logged in
<0.047<2 XSERVERINFO MANAGEACCOUNTURL MANAGELISTSURL MANAGEFILTERSURL
>0.003>* XSERVERINFO MANAGEACCOUNTURL {67}
http://redb@cuisine.blue.planet.com:800/bin/user/admin/bin/enduser
MANAGELISTSURL NIL MANAGEFILTERSURL NIL
2 OK Completed
<0.046<3 select "INBOX"
>0.236>* FLAGS (\Answered flagged draft deleted \Seen $MDNSent Junk)
* OK [PERMANENTFLAGS (\Answered flag draft deleted \Seen $MDNSent Junk
\*)]
* 1538 EXISTS
* 0 RECENT
* OK [UNSEEN 23]
* OK [UIDVALIDITY 1046219200]
* OK [UIDNEXT 1968]
3 OK [READ-WRITE] Completed
<0.045<4 UID fetch 1:* (FLAGS)
>0.117>* 1 FETCH (FLAGS (\Seen) UID 330)
* 2 FETCH (FLAGS (\Seen) UID 331)
* 3 FETCH (FLAGS (\Seen) UID 332)
* 4 FETCH (FLAGS (\Seen) UID 333)
* 5 FETCH (FLAGS (\Seen) UID 334)
<etc>
```

You can gather command telemetry that does not include end-user information by using the `imap.logcommands mschonfig` option (or in legacy configuration `local.imap.logcommands`). See *Messaging Server Administration Reference* for additional information.

To disable the telemetry logging, move or remove the directory that you created.

Checking stored Processes

The `stored` function performs a variety of important tasks such as deadlock and transaction operations of the message database, enforcing aging policies, and expunging and erasing messages stored on disk. If `stored` stops running, Messaging Server will eventually run into problems. If `stored` does not start when `start-msg` is run, no other processes will start.

- Check that the `stored` process is running. Run `imcheck`.
- Check for the log file build up in *store_root*`/mboxlist`.
- Check for `stored` messages in the default log file *msg-svr-base*`/log/default/default`
- Check that the time stamps of the following files (in directory *msg-svr-base*`/config/`) are updated whenever one of the following functions are attempted by the `stored` process:

stored Operations

stored Operation	Function
`stored.ckp`	Touched when a database checkpoint was initiated. Stamped approximately every 1 minute.
`stored.lcu`	Touched at every database log cleanup. Time stamped approximately every 5 minutes.
`stored.per`	Touched at every spawn of peruser db write out. Time stamped once an hour.

For more information on the `stored` process, see the `stored` page. For additional information on monitoring the `stored` function, see Monitoring `stored`

Checking Database Log Files

Database log files refer to sleepycat transaction checkpointing log files (in directory *store_root* `/mboxlist`). If log files accumulate, then database checkpointing is not occurring. In general, there are two or three database log files during a single period of time. If there are more files, it could be a sign of a problem.

Checking User Folders

If you want to check the user folders, you might run the command `reconstruct -r -n` (recursive no fix) which will review any user folder and report errors. For more information on the `reconstruct` command, see Repairing Mailboxes and the Mailboxes Database.

Checking for Core Files

Core files only exist when processes have unexpectedly terminated. It is important to review these files, particularly when you see a problem in the message store. On Oracle Solaris, use `coreadm` to configure `core` file location.

Monitoring imapd, popd and httpd

These processes provide access to IMAP, POP and Webmail services. If any of these is not running or not responding, the service will not function appropriately. If the service is running, but is over loaded, monitoring will allow you to detect this and configure it more appropriately.

Topics in this section:

- Symptoms of imapd, popd and httpd Problems
- To Monitor imapd, popd and httpd

Symptoms of imapd, popd and httpd Problems

Connections are refused or system is too slow to connect. For example, if IMAP is not running and you try to connect to IMAP directly you will see something like this:

```
telnet 0 143 Trying 0.0.0.0... telnet: Unable to connect to remote host:
Connection refused
```

If you try to connect with a client, you will get a message such as:

“Client is unable to connect to the server at the location you have specified. The server may be down or busy.”

To Monitor imapd, popd and httpd

- Can be monitored with `watcher` and `msprobe`. See Automatic Restart of Failed or Unresponsive Services and Monitoring Using msprobe and watcher Functions.
- Can be monitored with SNMP.
  If you have the SNMP set up, this is a very good way to monitor these processes. See SNMP Support . The server information is in the Network Services Monitoring MIB.
- Check log files.
  Look in the directory *msg-svr-base*/`log`/*service* where _ service_ can be http or IMAP or POP. In that directory you will find a number of log files. One filename is the name of the *service* (imap, pop, http) and the others are the name of the service plus a sequence number and a date concatenated to the service name. For example:
  `imap imap.29.1010221593 imap.31.1010394412 imap.33.1010567224`

The file with just the service name is the latest log. The other ones are ordered by the sequence number (here 29, 31, 33) and the one with the highest sequence number is the next newest one. (See Message Store Logging.)

If a server was shut down you might see something like this:

imap.12.1065431243:[07/Oct/2003:01:15:43 -0700] gotmail-2 imapd[20525]: General Warning: Sun Java System Messaging Server IMAP4 6.1 (built Sep 24 2003) shutting down

- Can be checked with `counterutil`. See the counterutil page and To Gather Message Store Counter Statistics by Using counterutil.
- Run the platform-specific command to verify that the imapd, popd and httpd processes are running. For example, in Oracle Solaris you can use the `ps` command and look for `imapd`, `popd` and `mshttpd`.
- You can set alarms for specified server performance thresholds by setting the server response configuration parameters described in Alarm Messages.
- See immonitor-access.

Monitoring `stored`

`stored` performs a variety of important tasks such as deadlock and transaction operations of the message database, enforcing aging policies, and expunging and erasing messages stored on disk. If `stored` stops running, the messaging server will eventually run into problems. If `stored` does not start when `start-msg` is run, no other processes will start. See the `stored` page for more information.

Topics in this section:

- Symptoms of stored Problems
- To Monitor `stored`

Symptoms of stored Problems

There are no outward symptoms.

To Monitor `stored`

- Check that the `stored` process is running. `stored` creates and updates a `pid` file in *msg-svr-base*`/data/proc` called `store`. The `pid` file shows an `init` state when recovering and a `ready` state when ready. For example:

```
231: cat store
28250
ready
```

The number on the first line is the process ID of `stored`.

```
232: ps -eaf | grep stored
inetuser 28250 1 0 Jan 05 ? 8:44
/opt/SUNWmsgsr/lib/stored -d
```

- Check for log file build up in *msg-svr-base*`/store/mboxlist`. Note that not every log file build up is caused by direct `stored` problems. Log files may also build up if `imapd` dies or there is a database problem.
- Check the timestamp on the following files in *msg-svr-base*`/config`:
  `stored.ckp` - Touched when attempt at checkpointing is made. Should get time stamped every 1 minute `stored.lcu` - Touched at every db log cleanup. Should get time stamped every 5 minutes `stored.per` - Touched at every spawn of peruser db writeout. Should get time stamped every 60 minutes
- Check for `stored` messages in the default log file *msg-svr-base*`/log/default/default`
- Can be monitored with `watcher` and `msprobe`. See Automatic Restart of Failed or Unresponsive Services and Monitoring Using msprobe and watcher Functions.

Monitoring the State of Message Store Database Locks

The state of database-locks is held by different server processes. These database locks can affect the performance of the message store. In case of deadlocks, messages will not be getting inserted into the store at reasonable speeds and the `ims-ms` channel queue will grow larger as a result. There are legitimate reasons for a queue to back up, so it is useful to have a history of the queue length in order to diagnose problems.

Topics in this section:

- Symptoms of Message Store Database Lock Problems
- To Monitor Message Store Database Locks

Symptoms of Message Store Database Lock Problems

Number of transactions are accumulating and not resolving.

To Monitor Message Store Database Locks

Use the command `imcheck -s` (used to be `counterutil -o db_lock`).

To Monitor Mailbox Quotas and Usage

You can monitor mailbox quota usage and limits by using the `imquotacheck` utility. The `imquotacheck` utility generates a report that lists defined quotas and limits, and provides information on quota usage.

For example, the following command lists all user quota information:

```
% imquotacheck
-------------------------------------------------------------------------
Domain red.siroe.com (diskquota = not set msgquota = not set) quota usage
-------------------------------------------------------------------------
diskquota         size(K)    %use    msgquota      msgs    %use    user
# of domains = 1
# of users = 705
no quota          50418             no quota      4392             ajonk
no quota              5             no quota      2                andrt
no quota         355518             no quota      2500             ansri
 ...
```

The following example shows the quota usage for user sorook:

```
% imquotacheck -u sorook
-------------------------------------------------------------------------
quota usage for user sorook
-------------------------------------------------------------------------
diskquota      size(K)    %use    msgquota      msgs     %use    user
no quota       1487               no quota      305              sorook
```

To list the usage of all users whose quota exceeds the least threshold in the rule file:
`imquotacheck`

To list quota information for a the domain `siroe.com`:
`imquotacheck -d siroe.com`

To send a notification to all users in accordance to the default rule file:
`imquotacheck -n`

To send a notification to all users in accordance to a specified *rulefile*, *myrulefile*, and to a specified mail template file, *mytemplate.file* (for more information, refer to imquotacheck page):
`imquotacheck -n -r` *myrulefile* `-t` *mytemplate.file*

To list per folder usages for one user user1 (will ignore the rule file):
`imquotacheck -u user1 -e`

Note
Certain functions have changed in `imquotacheck`. (Starting in Messaging Server 6, `imquotacheck` utility has superseded the `quotacheck` utility.) In Messaging Server 5, when you used the `quotacheck` utility to retrieve a list of users, `quotacheck` searched the local `mboxlist` database. This function duplicated the search function in the `mboxutil` utility.

Starting in Messaging Server 6, this duplicate function was removed from the `imquotacheck` utility. If you perform a user search with `imquotacheck`, the search is performed against the LDAP directory, not the local `mboxlist` database. To retrieve a list of users from the local `mboxlist` database, use the `mboxutil` utility.

To Monitor Message Store Database Statistics with imcheck

Use `imcheck -s` to monitor database statistics including logs and transactions. See imcheck.

Gathering Message Store Counter Statistics by Using counterutil

Topics in this section:

- To Get a Current List of Available Counter Objects
- counterutil Output
- Gathering Alarm Statistics by Using counterutil
- IMAP, POP, and HTTP Connection Statistics by Using counterutil
- Disk Usage Statistics by Using counterutil
- Server Response Statistics

To Get a Current List of Available Counter Objects

This utility provides statistics acquired from different system counters. (See the `counterutil` page.)

Here is how to get a current list of available counter objects:

```
# /opt/SUNWmsgsr/sbin/counterutil -l
Listing registry (/opt/SUNWmsgsr/data/counter/counter)
numobjects = 11
refcount = 1
created = 25/Sep/2003:02:04:55 -0700
modified = 02/Oct/2003:22:48:55 -0700
  alarm
  diskusage
  serverresponse
  imapstat
  httpstat
  popstat
  cgimsg
```

Each entry represents a counter object and supplies a variety of useful counts for this object. In this section we will only be discussing the `alarm`, `diskusage`, `serverresponse`, `popstat`, `imapstat`, and `httpstat` counter objects. For details on `counterutil` command usage, refer to the counterutil page.

counterutil Output

`counterutil` has a variety of flags. A command format for this utility may be as follows:

`counterutil -o` *CounterObject* `-i 5 -n 10`
where,
`-o` *CounterObject* represents the counter object `alarm`, `diskusage`, `serverresponse`, `popstat`, `imapstat`, and `httpstat`.
`-i 5` specifies a 5 second interval.
`-n 10` represents the number of iterations (default: infinity).
An example of `counterutil` usage is as follows:

```
# counterutil -o imapstat -i 5 -n 10
Monitor counteroobject (imapstat)
registry /gotmail/iplanet/server5/msg-gotmail/counter/counter opened
counterobject imapstat opened
count = 1 at 972082466 rh = 0xc0990 oh = 0xc0968
global.currentStartTime [4 bytes]: 17/Oct/2000:12:44:23 -0700
global.lastConnectionTime [4 bytes]: 20/Oct/2000:15:53:37 -0700
global.maxConnections [4 bytes]: 69
global.numConnections [4 bytes]: 12480
global.numCurrentConnections [4 bytes]: 48
global.numFailedConnections [4 bytes]: 0
global.numFailedLogins [4 bytes]: 15
global.numGoodLogins [4 bytes]: 10446
...
```

Gathering Alarm Statistics by Using counterutil

These alarm statistics refer to the alarms sent by `stored`. The alarm counter provides the following statistics:

counterutil alarm Statistics

Suffix	Description
`alarm.countoverthreshold`	Number of times crossing threshold.
`alarm.countwarningsent`	Number of warnings sent.
`alarm.current`	Current monitored valued.
`alarm.high`	Highest ever recorded value.
`alarm.low`	Lowest ever recorded value.
`alarm.timelastset`	The last time current value was set.
`alarm.timelastwarning`	The last time warning was sent.
`alarm.timereset`	The last time reset was performed.
`alarm.timestatechanged`	The last time alarm state changed.
`alarm.warningstate`	Warning state (yes(1) or no(0)).

IMAP, POP, and HTTP Connection Statistics by Using counterutil

To get information on the number of current IMAP, POP, and HTTP connections, number of failed logins, total connections from the start time, and so forth, you can use the command `counterutil -o` *CounterObject* `-i 5 -n 10`. Where *CounterObject* represents the counter object `popstat`, `imapstat`, or `httpstat`. The meaning of the `imapstat` suffixes is shown in the following table. The `popstat` and `httpstat` objects provide the same information in the same format and structure.

counterutil imapstat Statistics

Suffix	Description
currentStartTime	Start time of the current IMAP server process.
lastConnectionTime	The last time a new client was accepted.
maxConnections	Highest recorded number of concurrent TCP connections handled by IMAP server since the last counter reset.
numConnections	Total number of TCP connections successfully accepted by the current IMAP server. numConnections can include failed connections, but not always.
numCurrentConnections	Current number of active TCP connections.

numFailedConnections	Total number of failed TCP connections by the current IMAP server. This number accumulates until the server restart or reset by counterutil. numFailedConnections counts connections abnormally terminated, including unsuccessful accepts and connections successfully accepted but which had an error later. An error message is logged when a connection failed with an expected error. You can check your IMAP log files for error messages such as the following:
	`Unable to accept client connection: <error message>`
	`Socket error : <error message>`

numFailedLogins	Number of failed system logins served by the current IMAP server.
numGoodLogins	Number of successful system logins served by the current IMAP server.

Disk Usage Statistics by Using counterutil

The command `counterutil -o diskusage` generates following information:

counterutil diskusage Statistics

Suffix	Description
diskusage.availSpace	Total space available in the disk partition.
diskusage.lastStatTime	The last time statistic was taken.
diskusage.mailPartitionPath	Mail partition path.
diskusage.percentAvail	Disk partition space available percentage.
diskusage.totalSpace	Total space in the disk partition.

Server Response Statistics

The command `counterutil -o serverresponse` generates following information. This information is useful for checking if the servers are running, and how quickly they're responding.

counterutil serverresponse Statistics

Suffix	Description
`http.laststattime`	Last time http server response was checked.
`http.responsetime`	Response time for the http.
`imap.laststattime`	Last time imap server response was checked.
`imap.responsetime`	Response time for the imap.
`pop.laststattime`	Last time pop server response was checked.
`pop.responsetime`	Response time for the pop.

Monitoring User Access to the Message Store

Monitoring User Access to the Message Store

Messaging Server provides the command, `imsconnutil`, which allows you to monitor user's message store access via IMAP, POP and http. You can also determine the last log in and log out of users. This command works on a per message store basis and will not work across message stores.

> **Note**
> Use of this function or other Messaging Server functions to monitor, read or otherwise access user's email may constitute a potential source of liability if used in violation of applicable laws or regulations or if used in violation of the customer's own policies or agreements.

This command requires root access by the system user (default: `mailsrv`), and you must set the configuration variables `local.imap.enableuserlist`, `local.http.enableuserlist`, `local.enablelastaccess` to 1.

To list users currently logged on via IMAP or any web mail client, use the following command:

`imsconnutil -c`

To list the last IMAP, POP, or Messenger Express access (log in and log out) of every user on the message store use:

`imsconnutil -a`

The following command does two things: 1) it determines whether the specified user is currently logged on via IMAP or Messenger Express or any client that connects via mshttp (note that this does not work for POP because POP users generally do not stay connected), and 2) it lists the last time the users have logged on and off:

`imsconnutil -c -a -u` *user_ID*

Note that a list of users can be input from a file, one user per line, using the following command:

`imsconnutil -c -a -f` *filename*

You can also specify a particular service (`imap` or `http`) using the `-s` flag. For example, to list whether a particular user ID is logged onto IMAP or not, use the following command:

`imsconnutil -c -s imap -u` *user_ID*

Note that the `-k` option may only work if IMAP IDLE is configured. For a complete description of the `imsconnutil` syntax, refer to imsconnutil in Sun Java System Messaging Server 6.3 Administration Reference.

Here is some example output:

```
$ ./imsconnutil -a -u soroork
UID     IMAP last access    HTTP last access    POP last access
================================================================================
ed   08/Jul/2003:10:49:05   10/Jul/2003:14:55:52  ---NOT-RECORDED---
$ ./imsconnutil -c
IMAP
UID    TIME                AUTH            TO                  FROM
================================================================================
ed   17/Jun/2003:11:24:03  plain     172.58.73.45:193   129.157.12.73:2631
bil  17/Jun/2003:04:28:43  plain     172.58.73.45:193   129.158.16.34:2340
mia  17/Jun/2003:09:36:54  plain     172.58.73.45:193   192.18.184.103:3744
jay  17/Jun/2003:05:38:46  plain     172.58.73.45:193   129.159.18.123:3687
pau  17/Jun/2003:12:23:28  plaintext 172.58.73.45:193   192.18.194.83:2943
ton  17/Jun/2003:05:38:46  plain     172.58.73.45:193   129.152.18.123:3688
ani  17/Jun/2003:12:26:40  plaintext 172.58.73.45:193   192.18.164.17:1767
ani  17/Jun/2003:12:25:17  plaintext 172.58.73.45:193   129.150.17.34:3117
jac  17/Jun/2003:12:26:32  plaintext 172.58.73.45:193   129.150.17.34:3119
ton  17/Jun/2003:12:25:32  plaintext 172.58.73.45:193   192.18.148.17:1764
================================================================================
10 users were logged in to imap.
Feature is not enabled for http.
--------------------------------------------------------------------------------
```

Protecting Mailboxes from Deletion or Renaming

To Protect Mailboxes from Deletion or Renaming Except by an Administrator

You might want to protect some mailboxes from deletion or modification except by the administrator. The following procedure describes how to do this.

If someone other than an administrator attempts to delete, modify, or rename a protected mailbox, the error message `mailbox is pinned` is displayed.

- Set the `local.store.pin` {configutil) option by using the following format:

  ```
  configutil -o local.store.pin -v "mailbox1%mailbox2%mailbox 3"
  ```

  where *mailbox1*, *mailbox2*, and *mailbox 3* are the mailboxes to be protected (you can use spaces in mailbox names), and % is the separator between each mailbox.

Quotas & Automatic Mail Deletion

Message Store Quotas and Automatic Mail Deletion

- Quota Overview
- Setting Up Quotas
- Automatic Message Removal (Overview)
- Configuring Message Expiration

Also see Message Store Maintenance Queue for details of how expunged messages are removed.

Reducing Message Store Size Due to Duplicate Storage

Reducing Message Store Size Due to Duplicate Storage of Identical Messages

When a message is sent to multiple recipients, that message is placed in each recipient's mailbox. Some messaging systems store separate copies of the same message in each recipient's mailbox. By contrast, the Oracle Communications Messaging Server strives to retain a single copy of a message regardless of the number of mailboxes in which that message resides. It does this by creating hard links to that message in the mailboxes containing that message.

When other messaging systems are migrated to the Messaging Server, these multiple message copies may be copied over with the migration process. With a large message store, this means that a lot of messages are duplicated unnecessarily. In addition, multiple copies of the same message can be accumulated in normal server operation, for example, from IMAP append operations or other sources.

Messaging Server provides a new command called relinker that removes the excess message copies and replaces them with hard links to a single copy.

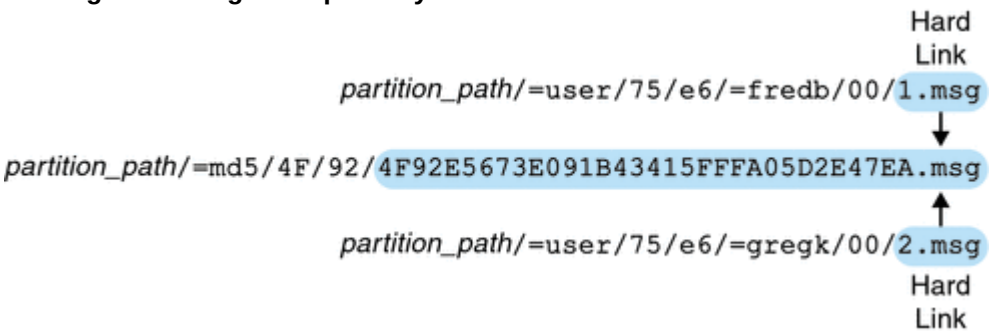
Topics:

Relinker Theory of Operations

The relinking function can be run in the command or realtime mode. When the relinker command is run, it scans through the message store partitions, creates or updates the MD5 message digest repository (as hard links), deletes excess message files, and creates the necessary hard links.

The digest repository consists of hard links to the messages in the message store. It is stored in the directory hierarchy *partition_path*/=md5. This directory is parallel to the user mailbox hierarchy _ partition_path_/=user (see Message Store Directory Layout). Messages in the digest repository are uniquely identified by their MD5 digest. For example, if the digest for fredb/00/1.msg is 4F92E5673E091B43415FFFA05D2E47, then *partition*/=user/*hashdir*/*hashdir*/=fredb/00/1.msg is linked to *partition*/=md5/_ hashdir_/*hashdir*/4F92E5673E091B43415FFFA05D2E47EA.msg. If another mailbox has this same message, for example, *partition_path*/=user/*hashdir*/*hashdir*/gregk/00/17.msg, that message will also be hard linked to *partition_path* /=md5/4F/92/4F92E5673E091B43415FFFA05D2E47EA.msg. This is shown in the following Figure.

Message Store Digest Repository



For this message, the link count will be three. If both messages are deleted from the mailboxes of fredb

and gregk, then the link count will be one and the message can be purged.

The relinker process can also be run in the realtime mode for similar functionality. See Using Relinker in the Realtime Mode for details.

Using `relinker` in the Command Line Mode

`relinker` scans through a message store partition, creates or updates the MD5 message repository (as hard links) and deletes excess message files. After relinker scans a store partition, it outputs statistics on the number of unique messages and size of the partition before and after relinking. To run more quickly on an already hashed store, relinker only computes the digest of the messages not yet present in =md5. It also has an option to erase the entire digest repository (which doesn't affect the user mailboxes).

The syntax for the command is as follows:

`relinker` [`-p` *partitionname*] [-d]

where *partitionname* specifies the partition to be processed (default: all partitions) and -d specifies that the digest repository be deleted. Sample output is shown below:

```
# relinker
Processing partition: primary
Scanning digest repository...
Processing user directories.............................
------------------------------------------------------------
Partition statistics           Before            After
------------------------------------------------------------
Total messages                 4531898         4531898
Unique messages                4327531         3847029
Message digests in repository        0         3847029
Space used                       99210Mb         90481Mb
Space savings from single-copy    3911Mb         12640Mb
------------------------------------------------------------
# relinker -d
Processing partition: primary
Purging digest repository...
------------------------------------------------------------
Partition statistics                Before         After
------------------------------------------------------------
Message digests in repository       3847029             0
------------------------------------------------------------
```

Relinker can take a long time to run, especially for the first time if there are no messages are in the repository. This is because it has to calculate the digest for every message (if the `relinker` criteria is configured to include all messages—see Configuring Relinker for information on configuring relinker criteria.) For example, it could take six hours to process a 100 Gigabyte message store. However, if run-time relinking is enabled see Using Relinker in the Realtime Mode.

If the `relinker` command line mode is used exclusively, and not the run-time option, it is necessary to purge the digest repository (`=md5`), otherwise messages purged in the store (`=user`) will not become available disk space since they still have a link in the digest repository (they become orphaned). If you are just performing a one-time optimization of the store—for example after a migration—you can run `relinker` once, then delete the entire repository with `relinker -d`. For repeated purging during migration, it is sufficient to just run the `relinker` command repeatedly, since each time it runs it also purges the expired or orphaned messages from the repository.

It is safe to run multiple instances of relinker in parallel with each processing a different partition (using

the `-p` option). Messages are only relinked inside the same partition.

Using Relinker in the Realtime Mode

The relinker function can be enabled in the realtime mode by setting the `configutil` parameter `local.store.relinker.enabled` to `yes`. Using relinker in the realtime mode will compute the digest of every message delivered (or restored, IMAP appended, and so forth) which matches the configured relinker criteria (Configuring Relinker), then look in the repository to see if that digest is already present. If the digest is present, it creates a link to it in the destination mailbox instead of creating a new copy of the message. If there is no digest, it creates the message and adds a link to it in the repository afterwards.

`stored` scans the digest repositories of each partition and purges the messages having a link count of 1, or which don't match the relinker criteria. The scan is done one directory at a time over a configurable time period. This is so that the I/O load is evenly distributed and doesn't noticeably impact other server operations. By default the purge cycle is 24 hours, which means messages can still be present on the disk for up to 24 hours after they've been deleted from the store or have exceeded the configured maximum age. This task is enabled when the relinker realtime mode is enabled.

Configuring Relinker

The following table shows the parameters used to set relinker criteria.

relinker configutil Parameters

Parameter	Description
`local.store.relinker.enabled`	Enables real-time relinking of messages in the append code and `stored` purge. The `relinker` command-line tool may be run even if this option is off. However since `stored` will not purge the repository, `relinker -d` must be used for this task. Turning this option on affects message delivery performance in exchange for the disk space savings. Default: 0
`local.store.relinker.maxage`	Maximum age in hours for messages to be kept in the repository, or considered by the `relinker` command-line. -1 means no age limit, that is, only purge orphaned messages from the repository. For `relinker` it means process existing messages regardless of age. Shorter values keep the repository smaller thus allow `relinker` or `stored` purge to run faster and reclaim disk space faster, while longer values allow duplicate message relinking over a longer period of time, for example, when users copy the same message to the store several days apart, or when running a migration over several days or weeks. Default: 24
`local.store.relinker.minsize`	Minimum size in kilobytes for messages to be considered by run-time or command-line `relinker`. Setting a non-zero value gives up the relinker benefits for smaller messages in exchange for a smaller repository. Default: 0
`local.store.relinker.purgecycle`	Approximate duration in hours of an entire `stored` purge cycle. The actual duration depends on the time it takes to scan each directory in the repository. Smaller values will use more I/O and larger values will not reclaim disk space as fast. 0 means run purge continuously without any pause between directories. -1 means don't run purge in `stored` (then purge must be performed using the `relinker -d` command). Default: 24

Specifying Administrator Access to the Message Store

Specifying Administrator Access to the Message Store

This information describes how to grant store privileges to the message store for your Oracle Communications Messaging Server installation. See Managing Message Store Partitions and Adding Storage for conceptual information.

Topics:

Overview

Message store administrators can view and monitor user mailboxes and specify access control for the message store. Store administrators have proxy authentication privileges to any service (POP, IMAP, HTTP, or SMTP), which means they can authenticate to any service using the privileges of any user. These privileges allow store administrators to run certain utilities for managing the store. For example, using `MoveUser`, store administrators can move user accounts and mailboxes from one system to another.

> **Note**
> Other users might also have administrator privileges to the store. For example, some administrators may have these privileges.

See also Protecting Mailboxes from Deletion or Renaming.

To Add an Administrator Entry

- To add an administrator entry at the command line:

  ```
  configutil -o store.admins -v "adminlist"
  ```

  where *adminlist* is a space-separated list of administrator IDs. If you specify more than one administrator, you must enclose the list in quotes. In addition, the administrator must be a member of the Service Administrator Group (in the LDAP user entry: `memberOf: cn=Service Administrators,ou=Groups,o=usergroup`). You must restart `imapd` for the system to recognize the change in `store.admins`.

To Modify an Administrator Entry

- To modify an existing entry in the message store Administrator UID list at the command line:

  ```
  configutil -o store.admins -v "adminlist"
  ```

  where *adminlist* is a space-separated list of administrator IDs. If you specify more than one administrator, you must enclose the list in quotes. In addition, the administrator must be a member of the Service Administrator Group (in the LDAP user entry: `memberOf: cn=Service Administrators,ou=Groups,o=usergroup`).

You must restart `imapd` for the system to recognize the change in `store.admins`.

To Delete an Administrator Entry

- To delete store administrators at the command line, you can edit the administrator list as follows:

  `configutil -o store.admins -v "`*adminlist*`"`

  where *adminlist* is a space-separated list of administrator IDs. If you specify more than one administrator, you must enclose the list in quotes. In addition, the administrator must be a member of the Service Administrator Group (in the LDAP user entry: `memberOf: cn=Service Administrators,ou=Groups,o=usergroup`).

  You must restart `imapd` for the system to recognize the change in `store.admins`.

Troubleshooting the Message Store

Troubleshooting the Message Store

This information provides guidelines for troubleshooting your message store as well as recovery procedures for when the message store becomes corrupted or unexpectedly shuts down.

Topics:

See also:

Repairing Mailboxes and the Mailboxes Database (reconstruct Command)

If one or more mailboxes become corrupt, use the `reconstruct` utility to rebuild the mailboxes or the mailbox database. You can use this utility to recover from almost any form of data corruption in the mail store. See Error Messages Signifying that `reconstruct` is Needed and the `reconstruct man page` for more details.

Reduced Message Store Performance

Message store problems can occur if the `mboxlist` database cache is too small. Specifically, Message store performance can slow to unacceptable levels and can even dump core. Refer to the topic on performance tuning considerations for a Messaging Server architecture in *Unified Communications Suite Deployment Planning Guide*.

Red Hat Linux - Messaging Server Patch 120230-08 IMAP, POP and HTTP Servers Not Starting Due to Over Sessions Per Process

After installing this patch, when you try to start Messaging Server, the IMAP, POP and HTTP servers do not start and may send the following example error logs:

```
http server - log:
[29/May/2006:17:44:37 +051800] usg197 httpd[6751]: General Critical: Not
enough file
descriptors to support 6000 sessions per process; Recommend ulimit -n
12851 or 87
sessions per process.
pop server - log:
[29/May/2006:17:44:37 +051800] usg197 popd[6749]: General Critical: Not
enough file
descriptors to support 600 sessions per process; Recommend ulimit -n
2651 or 58
sessions per process.
Once these values setting in /opt/sun/messaging/sbin/configutil then
imap server
failed to start
imap server - log:
[29/May/2006:17:44:37 +051800] usg197 imapd[6747]: General Critical: Not
enough
file descriptors to support 4000 sessions per process; Recommend ulimit
-n 12851
or 58 sessions per process.
```

Set the appropriate number of file descriptors for all three server sessions. Additional file descriptors are available by adding a line similar to the following to the `/etc/sysctl.conf` file and using `sysctl -p` to reread that file:

```
fs.file-max = 65536
```

You must also add a line like the following to the `/etc/security/limits.conf` file:

```
*   soft  nofile  65536
*   hard  nofile  65536
```

Messenger Express or Communications Express Not Loading Mail Page

If the user cannot load any Messenger Express pages or the Communications Express mail page, the problem might be that the data is getting corrupted after compression. This can sometimes happen if the system has deployed a outdated proxy server. To solve this problem, try setting `local.service.http.gzip.static` and `local.service.http.gzip.dynamic` to `0` to disable data compression. If this solves the problem, you may want to update the proxy server.

Command Using Wildcard Pattern Does Not Work

Some UNIX shells may require quotes around wildcard parameters and some will not. For example, the C shell tries to expand arguments containing wild cards (*, ?) as files and will fail if no match is found. These pattern matching arguments may need to be enclosed in quotes to be passed to commands like `mboxutil`.

For example:

```
mboxutil -l -p user/usr44*
```

works in the Bourne shell, but fails with tsch and the C shell. These shells would require the following:

```
mboxutil -l -p "user/usr44*"
```

If a command using a wildcard pattern does not work, verify whether you need to use quotes around wildcards for that shell.

Unknown/invalid Partition

A user can get the message “Unknown/invalid partition” in Messenger Express if their mailbox was moved to a new partition that was just created and Messaging Server was not refreshed or restarted. This problem only occurs on new partitions. If you now add additional user mailboxes to this new partition, you will not have to do a refresh/restart of Messaging Server.

User Mailbox Directory Problems

A user mailbox problem exists when the damage to the message store is limited to a small number of users and there is no global damage to the system. The following guidelines suggest a process for identifying, analyzing, and resolving a user mailbox directory problem:

1. Review the log files, the error messages, or any unusual behavior that the user observes.
2. To keep debugging information and history, copy the entire *store_root*`/mboxlist/` user directory to another location outside the message store.
3. To find the user folder that might be causing the problem, run the command `reconstruct -r -n`. If you are unable to find the folder using `reconstruct`, the folder might not exist in the `folder.db`.
   If you are unable to find the folder using the `reconstruct -r -n` command, use the `hashdir` command to determine the location.
4. Once you find the folder, examine the files, check permissions, and verify the proper file sizes.
5. Use `reconstruct -r` (without the `-n` option) to rebuild the mailbox.
6. If `reconstruct` does not detect a problem that you observe, you can force the reconstruction of your mail folders by using the `reconstruct -r -f` command.
7. If the folder does not exist in the `mboxlist` directory (*store_root*`/mboxlist`), but exists in the `partition` directory *store_root*`/partition`), there might be a global inconsistency. In this case, you should run the `reconstruct -m` command.
8. If the previous steps do not work, you can remove the `store.idx` file and run the `reconstruct` command again.

   Caution
   You should only remove the `store.idx` file if you are sure there is a problem in the file that the `reconstruct` command is unable to find.

9. If the issue is limited to a problematic message, you should copy the message file to another location outside of the message store and run the command `reconstruct -r` on the `mailbox/` directory.
10. If you determine the folder exists on the disk (*store_root*`/partition/` directory), but is apparently not in the database (*store_root*`/mboxlist/` directory), run the command `reconstruct -m` to ensure message store consistency.

For more information on the `reconstruct` command, see Repairing Mailboxes and the Mailboxes Database (`reconstruct` Command).

Store Daemon Not Starting

If `stored` won't start and returns the following error message:

```
# <msg-svr-base>/sbin/start-msg
<msg-svr-base>: Starting STORE daemon ...Fatal error: Cannot find group in
name service
```

This indicates that the UNIX group configured in `local.servergid` cannot be found. `Stored` and others need to set their `gid` to that group. Sometimes the group defined by `local.servergid` gets inadvertently deleted. In this case, create the deleted group, add `mailsrv` to the group, change ownership of the *instance_root* and its files to `mailsrv` and the group.

User Mail Not Delivered Due to Mailbox Overflow

The message store has a hard limit of two gigabytes for a `store.idx` file, which is equivalent to about one million messages in a single mailbox (folder). If a mailbox grows to the point that the `store.idx` file will attempt to exceed two gigabytes, the user will stop receiving any new email. In addition, other processes that handle that mailbox, such as imapd, popd, mshttpd, could also experience degraded performance.

If this problem arises, you will see errors in `mail.log_current` such as this:

```
05-Oct-2005 16:09:09.63 ims-ms      Q 7 ... System I/O error.
Administrator, check server log for details. System I/O error.
```

In addition, the MTA log file will have an errors such as this:

```
[05/Oct/2005:16:09:09 +0900] jmail ims_master[20745]: Store Error:
Unable to append cache for user/admin: File too large
```

You can determine this problem conclusively by looking at the file in the user's message store directory, or by looking in the `imta` log file to see a more detailed message.

The immediate action is to reduce the size of the file. Either delete some mail, or move some of it to another mailbox. You could also use `mboxutil -r` to rename the folder out of the way, or `mboxutil -d` to delete the folder (see mboxutil).

Long-term, you will need to inform the user of mailbox size limitations, implement an aging policy (see Configuring Message Expiration), a quota policy (see Message Store Quota (Overview)), set a mail box limit by setting `local.store.maxmessages` (see Messaging Server Configuration), set up an archiving system, or do something to keep the mailbox size under control.

IMAP Events Become Slow

Symptom: After working fine for a short period of time, many IMAP events become unreasonably slow, with some events taking over a second.

Diagnosis: You have the Event Notification Service (ENS) plugin, `libibiff`, configured, but ENS is not

running or not reachable. See Administering Event Notification Service in Messaging Server for ENS details.

Solution: If you want ENS notifications, make sure the ENS is enabled and configured correctly. If you do not want ENS notifications, make sure that `libibiff` is not being loaded. Typical bad configuration:

```
local.store.notifyplugin = /opt/sun/comms/messaging/lib/libibiff
local.ens.enable = 0
```

Use either of the following for solution configurations:

```
local.store.notifyplugin =
local.ens.enable = 0
```

or

```
local.store.notifyplugin = /opt/sun/comms/messaging/lib/libibiff
local.ens.enable = 1
```

Upgrading the Message Store

Upgrading the Message Store

This information describes the data components and tools of message store, and how they affect data upgrades. It provides the technical background for upgrade planning. If you are using the upgrade procedure described in the "Upgrading the Messaging Server" section in *Messaging Server 8.0 Installation and Configuration Guide*, you do not need to concern yourself with this level of message store technical detail. However, if you need to customize your message store upgrade process, use this information as a guideline.

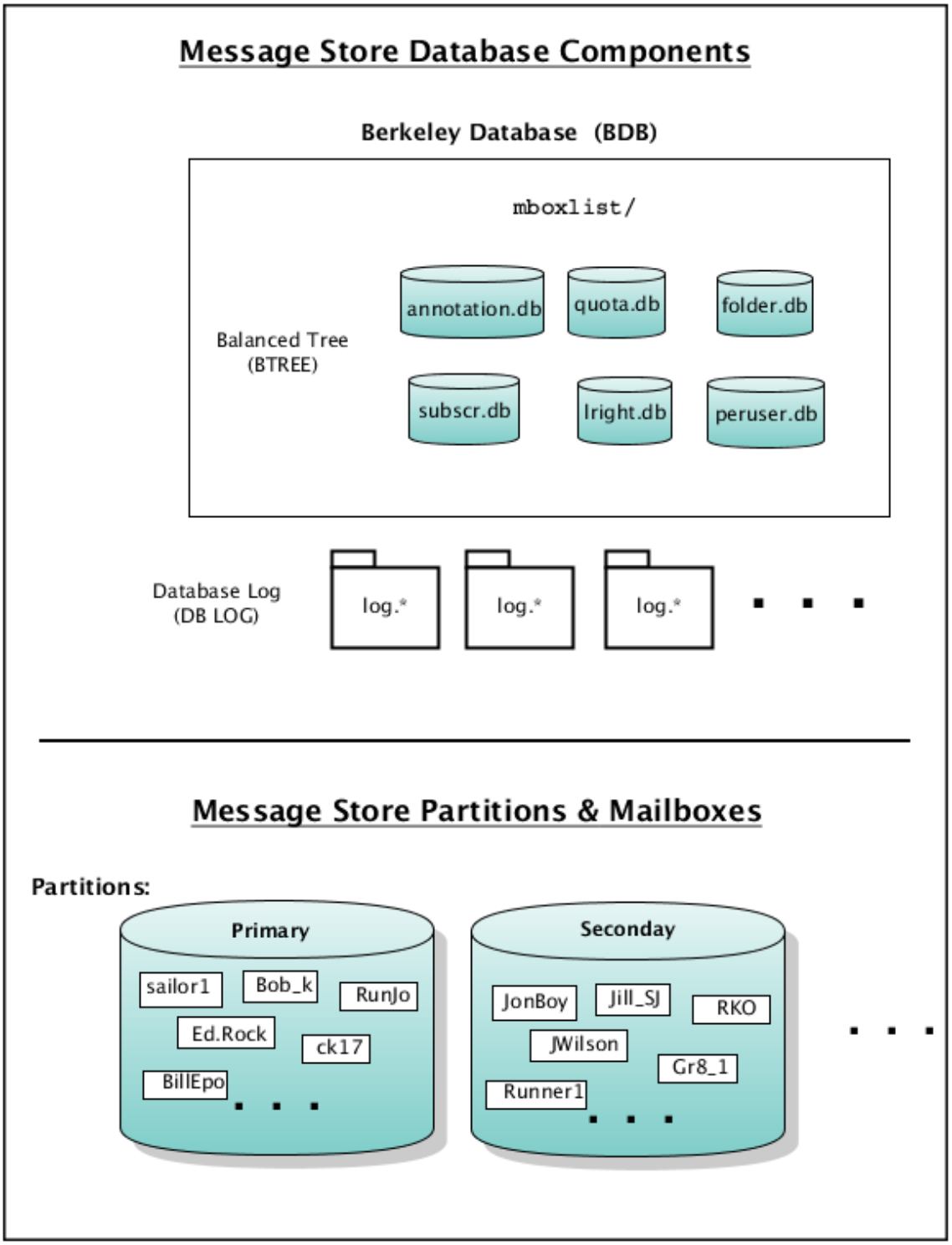
For specific information about significant design changes in the message store between versions that can impact disk I/O and performance, see Significant Changes in the Message Store Between Versions.

Topics:

Architecture and Components

The following figure shows the message store architecture and its components. See Message Store Directory Layout for a view of the message store file structure.

Message Store and Components



The versions of Messaging Server and its message store are: 4.15, 5.x, 6.0, 6.1, 6.2, 6.3, 7.0, 7.1, 7.2, 7.3, 7.4, and 7.5. During this time, there have been four *mailbox versions*: 1_1, 1_2, 1_3, and 1_4.

The message store uses the *Berkeley Database* provided by Sleepycat Software and, since 2006, Oracle Corporation. The message store database files are stored in a directory called `mboxlist` (see Message Store Directory Layout), and so it is often called the `mboxlist` database.

The message store has used a number of versions of the Berkeley Database over the course of its existence. Thus, when you upgrade your message store, the Berkeley Database may also be upgraded. The database engine upgrade has complexities and implications for the data upgrade. (These

complexities and implications are handled in the Message Store upgrade tools and instructions described in the "Upgrading the Messaging Server" section in|*Messaging Server 8.0 Installation and Configuration Guide*, but these details are described here for custom upgrade plans.)

The `mboxlist` database is stored in the *BTREE database files*. The BTREE database files store information about the message store users and mailboxes (see Message Store Directory Layout). The number and types of files, as well as the structure of files themselves have varied over the life of the Berkeley Database. The message store has used three different versions of the BTREE files. These are BTREE 6, 8 and 9.

The Berkeley Database is transactional, so each transaction is logged in a *database log file*. Database log files are used for recovery. All database changes are recorded in the database log file. When the server is crashed and restarted, it uses the log file to bring the database back to a consistent state. Before you do an upgrade, make sure you have a clean shutdown and recovery by running `stored -r`.

The format of the database log files has evolved over the years, and so the log files have different versions. Messaging Server has used five different log file versions: 2, 3, 8, 11, and 14.

Message Store Component Version Compatibilities

The following table contains the version number of various database components in the message store with respect to upgrade.

Message Store Database Components

Messaging Server	Mailbox	Berkeley Database	Database BTREE	Database Log Files
4.15	1_1	2.6	6	2
5.x	1_2	2.6	6	2
6.0	1_2	3.2.9	8	3
6.1	1_2	4.2	9	8
6.2	1_2	4.2	9	8
6.3	1_2	4.4	9	11
7.0	1_3	4.4	9	11
7 Update 1	1_3	4.7.25	9	14
7 Update 2	1_3	4.7.25	9	14
7 Update 3	1_3	4.7.25	9	14
7 Update 4	1_3	4.7.25	9	14
7 Update 5	1_4	5.3.21	9	19

In general, when you upgrade the messaging server from one version to another, you also have to upgrade the components that have a different version. Certain message components are not compatible with other components. The following section describes the various compatibilities and incompatibilities.

Mailbox Version Compatibilities

The message store upgrades mailboxes from version 1_1 to 1_2, from 1_2 to 1_3, and from 1_3 to 1_4 automatically. Usually, a mailbox is upgraded when the end users select their mailboxes, or when messages are delivered to the mailboxes after the message store software is upgraded. The message store checks the mailbox version in the mailbox header and upgrades the mailbox if needed.

Mailbox upgrade increases the load on the server. Disable the non-essential tasks during the transition period, such as:

- Disable peruser.db archive: `configutil -o local.store.seenckpinterval -v 0`
- Disable last access time tracking: `configutil -o local.enablelastaccess -v no`
- Decrease DB snapshot frequency
- Decrease log level
- Temporarily disable delivery
- Temporarily disable impurge

You can upgrade the mailboxes manually with `imcheck -H` during off-peak hours. The `imcheck` command opens every mailbox, which triggers the upgrade. You can run this command to make sure all the mailboxes are upgraded.

To upgrade from mail version 1_1 to 1_4, you have to use a migration tool such as `imsbackup` and `imsrestore`.

Downgrading mailbox versions is not automatic. You have to use a migration tool such as `imsbackup` and `imsrestore` to downgrade a mailbox.

Upgrading and Downgrading the Berkeley Database (BDB)

The Message Store uses the Berkeley Database to store various data. The email messages themselves are not part of the data stored in the BDB. If the database is upgraded to a version that is incompatible with the previous version, the database files will become incompatible with older versions of the Message Store. We recommend, therefore, that you make a backup copy of the database before the upgrade in case you want to back out of the upgrade.

The primary database which needs to be backed up is located by default in the `mboxlist/` directory, and consists of **`.db` files, and `log.`** files. The `__*` files are cache files for the database and do not need to be copied. A correct copy of the database ensures data is in a consistent state. The message store and utilities must be shut down. Running `stored -r` will make sure the cache files are synced to the database files. Both database and log files are required.

If you need to back out of a patch or update which upgrades the Berkeley Database to a version that is incompatible with previous versions and you did not make a backup, you may be able to rely on a previous database snapshot. Database snapshots are located by default in the `dbdata/` directory. A valid database snapshot directory will have a `.verified` file. The `.verified` file indicates that the snapshot has been recovered, verified, and is ready to be used.

Normally when the store is brought up, the `stored` process replaces the `mboxlist` directory with any snapshots needed. In the case where you have backed out of a database upgrade, `stored` may not take into account that a downgrade has occurred. It may therefore be necessary to replace the valid snapshot manually. To do this, move the current database files in `mboxlist/` out of the directory and move all the files from the chosen snapshot directory into the `mboxlist/` directory. Be sure to remove the `__*` tmp files as well. Note that if the store is configured with `store.dbtmpdir`, the tmp files will be in a different location.

If you have no database backup and no valid snapshots, it may be necessary to move the upgraded database out of the way, and rebuild it from scratch. Under normal circumstances, the Message Store rebuilds the database while allowing users to access their mail. Since doing this puts a heavier load on the system, you should create proper database backups instead.

Normal reparation of the database should be done after putting an older version in place by running the `reconstruct -m` and `reconstruct -r` commands.

Some of these manual requirements might be addressed in future releases.

Note that the Berkeley database consist of a number of BTREE files, LOG files and temporary files (tmp

file location is configurable with store.dbtmpdir). In order to upgrade the database, run `stored -r` before replacing the new libraries and binaries. Note that `stored -r` runs automatically during the proper upgrade process.

In the unlikely event that `stored -r` fails, check the store log files to determine the cause, run `stored` alone to perform a database recovery, and run `stored -r` again. In normal circumstances, this should not be a problem unless there is some underlying system problem which you should resolve before upgrading.

Database Btree File

BTREE version 8 and 9 are compatible. Upgrade is not needed.

To upgrade the BTREE files from version 6 (BDB 2.6) to a higher version, copy the database files from the old location (example: `/usr/iplanet/server5/msg-store/store/mboxlist` for 5.2) to the new `mboxlist` location (example: `/var/opt/SUNWmsgsr/store/mboxlist`), then run `ims_db_upgrade`.

Database Log Files

Database log files cannot be upgraded. When the log version changes, run the legacy version of `stored -r` to process the log files (recover the database) before upgrading. Do not remove the old log files.

The following message is logged when the server restart.

'Skipping log file...historic log version'

The store daemon creates a new log file with the new version.

IMAPD, MSHTTPD and Convergence

Starting in Messaging Server 6.3, the webmail server (`mshttpd`) uses `imapd` to access the Message Store. `imapd` and `mshttpd` in 6.3 and 7.*x* are fully compatible, so simultaneous upgrade is not required. To prevent memory problems due to redundant IMAP sessions from `mshttpd`, you should run with only one mshttpd process. If you serve a lot of concurrent webmail users, this might require an upgrade to 64-bit so that you have enough virtual address space for the process.

Convergence requires webmail server 7.*x*. To use Convergence, you must upgrade `mshttpd` to 7.*x*.

Upgrading From Messaging Server 32-bit to 64-bit

Run the legacy version of `stored -r` before upgrading the Messaging Server software from 32-bit to 64-bit.

If you run the 64-bit version, then it is more efficient to run with only one `mshttpd` process. See mshttpd Changes Starting with Messaging Server 6.3 for more information.

You might also want to reduce the number of `imapd` process. See Why Using Messaging Server 64-bit Edition Is Better.

> **Note**
> The Sun Java System Messaging Server 6.3 64-bit Installation Technical Note also has information about upgrading from 32-bit to 64-bit.

Migrating from x86 to SPARC

The message store data formats are architecture dependent. You cannot transfer any data components in the matrix from one architecture to another directly. To migrate the message store from an x86 machine to a SPARC machine (or from SPARC to x86), use `imsbackup` and `imsrestore` or `rehostuser`.

stored -r

`stored -r` performs a final recovery and cleanly removes the database temp files to prepare the database for an upgrade.

For example:

```
# /opt/SUNWmsgsr/lib/stored -r
removing mboxlist environment ... done
removing lock environment ... done
removing session db ... done
```

Make sure `stored -r` completes successfully.

The Messaging Server 6.2 `stored -r` command requires the watcher process. Make sure that the watcher is running before you perform `stored -r`.

The `stored -r` command was introduced in Messaging Server 6.1. Prior to Messaging Server 6.1, use the following procedure:

```
# stop-msg
# start-msg store
# stop-msg
# rm ${dataroot}/store/mboxlist/__db*
# rm ${dataroot}/store/mboxlist/folderlock/*
# rm ${dataroot}lock/__db*
```

ims_db_upgrade

`ims_db_upgrade` copies the database files to a backup directory, upgrades the database files to the current version and validates the new database. If upgrade is not successful, the backup files are restored. If the database is validated, the backup files are removed.

Downgrading

Mailbox and BDB downgrade are not supported. To downgrade a message store to an older version with incompatible mailboxes or databases, you must restore the mailboxes and mboxlist database from backup.

mshttpd Changes Starting with Messaging Server 6.3

The issue of having multiple `mshttpd` processes (or moreover, the recommendation to have only one `mshttpd` process) is not a Messaging Server 32-bit versus 64-bit issue. Starting with Messaging Server 6.3, configuring more than one `mshttpd` process now:

- Increases the load on the `imapd` processes by a factor equal to the number of `mshttpd` processes.
- Is not not as necessary as it used to be, and probably is no longer necessary at all.

Starting with Messaging Server 6.3, each mshttpd process can have a separate IMAP connection for each concurrent Webmail (or Communications Express or Convergence) session. Thus, you need to factor that into calculating your maxsessions and numprocesses for imapd.

Previously, the main reason for having more than one mshttpd process was the same as for imapd: large mailboxes run into the 32-bit address space limitation and the only solution was to spread sessions over more processes. For imapd, the solution is to upgrade to Messaging Server 64-bit Edition. However, for mshttpd, starting with Messaging Server 6.3, it is no longer the same issue as for imapd because mshttpd no longer accesses the message store files directly. Instead, mshttpd accesses the message store by using IMAP. Consequently, each mshttpd process may have an IMAP connection for each concurrent session.

Though it is hard to speculate just how much more or less memory that actually means, assume that even if you are running Messaging Server 32-bit (starting with Messaging Server 6.3) that you only need run only one mshttpd process. Then monitor the virtual memory size of the mshttpd process. If the size exceeds 3 Gbytes, you might need more than one mshttpd process. However, considering that multiple mshttpd processes increase the load on imapd, if your server cannot support the load in one 32-bit mshttpd process, you should upgrade to Messaging Server 64-bit Edition.

Significant Changes in the Message Store Between Versions

Significant Changes in the Message Store Between Versions

This document describes the significant design changes in the message store that can impact disk I/O and performance. Administrators and deployment designers may need to be aware of these changes for deployments where mboxlist, index, and message data are on different file systems or pools.

Topics:

Changes from Messaging Server 6.3 to Messaging Server 7.0

The following changes were made between Messaging Server 6.3 and Messaging Server 7.0:

Changes to `store.idx`

The `store.idx` file is separated into two or more files. The `store.idx` file contains the mailbox's meta data and a 128 bytes fixed length record for every message in the folder. The `store.cN` files contain the variable length cache records. The average cache record size is approximately 2 kilobytes.

Benefits

- Mailbox expunge is much faster
- Supports very large mailboxes (up to 33554431 messages)

I/O Impact

- More files (inodes) on the index partitions
- More I/O for very small mailboxes
- Initial mailbox access triggers mailbox upgrade automatically

Message Store Maintenance Queue and `impurge`

A BDB queue is added to the message store to manage maintenance tasks. Cache file purge and message file cleanup tasks are queued and executed by `impurge` which runs continuously.

Benefits

- Less I/O overall (especially write)
- Event driven message store maintenance
- Higher performance
- Eliminates sudden increase in load during nightly cleanup

I/O Impact

- Less write access
- Higher disk space usage (approximately 10%)

Mailbox Self-Healing (Auto-Repair)

Mailbox corruption is detected and scheduled for repair automatically by `impurge`.

Benefits

- Increases availability
- Reduces mailbox administration cost

I/O Impact

- Minor I/O increase due to error detection and repair.

Changes from Messaging Server 7 to Messaging Server 7 Update 1

The following changes were made between Messaging Server 7 and Messaging Server 7 Update 1:

- Berkeley Database Upgrade

Berkeley Database Upgrade

The Berkeley Database was upgrade from version 4.4 to version 4.7.25.

Benefits

- Higher throughput

I/O Impact

- None

Changes from Messaging Server 7 Update 1 to Messaging Server 7 Update 5

The following changes were made between Messaging Server 7 Update 1 and Messaging Server 7 Update 5:

- Changes to the Owner's Seen and Deleted Flags
- Immediate flag update and state sharing
- Change to the `service.imap.capability.condstore` parameter
- Changes to the Berkeley Database
- Changes to `mboxlist` and `lockdir` BDB environments

Changes to the Owner's Seen and Deleted Flags

The owner's seen and deleted flags have been moved from the Berkeley Database to the `store.idx` file.

Benefits

- Higher throughput by reducing Berkeley Database lock contentions
- Performance bottlenecks can be fixed by adding more spindles

I/O Impact

- Less I/O on the mboxlist file system
- More I/O on the index file system
- Initial mailbox access triggers mailbox upgrade automatically

Immediate flag update and state sharing

Flags and ACL changes are flushed to the disk and shared immediately across IMAP sessions. New message arrival are notified immediately.

Benefits

- IMAP sessions are more up-to-date

I/O Impact

- More I/O on the index file system

Change to the `service.imap.capability.condstore` parameter

The `service.imap.capability.condstore` parameter is enabled by default.

Benefits

- IMAP clients can utilize CONDSTORE
- Improve overall performance (when CONDSTORE is utilized)

I/O Impact

- Minimal (because flags and MODSEQ are updated together)

Changes to the Berkeley Database

The Berkeley Database was upgrade from version 4.7.25 to version 5.3.21. The default cache size increased from 16 MB to 64 MB. Moved the maintenance queue to the mboxlist environment.

Benefits

- Increase performance and scalability
- Simplify store maintenance tasks

I/O Impact

- A small increase in database environment (`tmp` file) size.

Changes to `mboxlist` and `lockdir` BDB environments

Mboxlist and lockdir DB environments (store.dbtmpdir and local.lockdir) default to `tmpfs`.

Benefits

- Better performance

I/O Impact

- Less I/O on the data root file system.

Chapter 21. Messaging Archiving

Message Archiving

This information describes archiving concepts for Messaging Server. It does not provide instructions on how to set up an archiving system.

Topics:

Microsoft Exchange Envelope Journaling

> The following feature was introduced in **Messaging Server 7.2**.

Messaging Server is able capture sieve action to produce Microsoft Exchange's "envelope journaling" format. This format consists of a multipart MIME message where the first part contains envelope information is a semi-structured format and the second part is the actual message. This new format by specifying a `:journal` parameter to capture:

```
capture :journal "trigger-address";
```

Archiving Overview

A message archiving system saves all or specified incoming and outgoing messages on a system separate from Messaging Server. Sent, received, deleted, and moved messages can all be saved and retrieve in an archive system. Archived messages cannot be modified or removed by email users, so the integrity of incoming and outgoing messages is maintained. Message archiving is useful for compliance record keeping, but it is also useful for message store management. For example, some customers may use archiving to perform message back-up or to move older messages from more expensive message store storage to less expensive archive storage.

Archived messages can be accessed through a separate archiving software GUI client or through Messaging Server. If the messages are deleted from Messaging Server, then the archiving client can be used to search for and retrieve those deleted messages since archived messages are never deleted. Note, however, that archived messages are not stored in mailbox folders as they are in the Messaging Server.

The system can also be set up so that archived messages can be accessed from Messaging Server. For example, you can set up a system to archive messages over 2 years old. Instead of having message bodies reside in the message store, they would instead reside in the archive system. From the users standpoint, the message appears no different from a regular email message. The same header and subject information will appear (this is still stored in the message store storage), but the message body is downloaded from the archive server by the message store when needed. Thus, there may be a slight delay as messages are downloaded from the archive server. In addition, archived messages cannot be searched from the email client. Searching must be done from the archiving GUI.

Message Archiving Systems: Compliance and Operational

There are two types of archiving, compliance and operational. Compliance archiving is used when you have a legal obligation to maintain strict retrievable email record keeping. Selected email (selected by user(s), domain, channel, incoming, outgoing and so on) coming into the MTA is copied to the archive system before being delivered to the message store or the internet. Archiving can be set to occur either before or after spam and virus filtering.

Operational archiving is used for mail management purposes. For example:

- To reduce storage usage on the Messaging Server message store by moving less used (older) messages to an archiving system which uses lower cost storage.
- As an alternative for data backup.

Note that compliance and operational archiving are not exclusive. That is, you can set up your system so that it does both compliance and operational archiving.

Message Archiving Using the Sun Compliance and Content Management Solution

Feature No Longer Available

This product is no longer available as the Sun Compliance and Content Management Solution. It is now marketed by Unify as Central Archive for Email. The documentation is offered for legacy systems. Support for Microsoft Envelope Journaling Format is provided with Messaging Server. See Messaging Archiving for more information.

Message Archiving Using the Sun Compliance and Content Management Solution

This technical note describes how to archive messages coming into and out of the Messaging Server using the Sun Compliance and Content Management Solution. Whether you require archiving for regulatory, compliance or litigation purposes, or you want to manage the growth of your message store and reduce the storage costs, the Sun Compliance and Content Management Solution can achieve this. At this time, the Sun Compliance and Content Management Solution uses the AXS-One Records Compliance Management application. The Sun Compliance and Content Management Solution refers to the entire solution including servers and applications. AXS-One is used to specify requirements for the actual archiving and portal applications.

This technical note describes how to configure the Messaging Server side. Refer to the AXS-One documentation or technical support for deployment and configuration on the AXS-One side. This Technical Note assumes that a trained technical team will be helping you to size, deploy and configure this system.

Topics:

Archiving Overview

A message archiving system saves a copy of incoming and outgoing messages on a system separate from Messaging Server. Sent, received, deleted, and moved messages can all be saved in, and retrieved from, an archive system. Archived messages cannot be modified or removed by email users so the integrity of incoming and outgoing messages is maintained. Message archiving is useful for compliance record keeping, message store management, and message back up.

Archived messages can be viewed through the AXS-One portal or through Messaging Server. If the messages are deleted from Messaging Server, the AXS-One portal can be used to retrieve those deleted messages. Note, however, that archived messages are not stored in a mailbox folders structure as they are in the Messaging Server. Refer to the AXS-One literature for details.

The system can also be set up so that Messaging Server displays archived messages. For example, you can set up your system to archive messages over two years old. When the user fetches the message from a client, the message store downloads it from the archive server and displays it. From the user's

point of view, the message looks the same as an normal email message. On the Messaging Server, however, instead of the message body, there is a **stub** that points to a URL of the contents of the message body stored on the archive system.

Message Archiving Systems: Compliance and Operational

There are two types of archiving, compliance and operational. Compliance archiving is used when you have a legal obligation to maintain strict retrievable email record keeping. Selected email (selected by user(s), domain, channel, incoming, outgoing and so on) coming into the MTA is copied to the archive system before being delivered to the message store or the internet. Archiving can be set to occur either before or after spam and virus filtering.

Operational archiving is used for mail management purposes. For example:

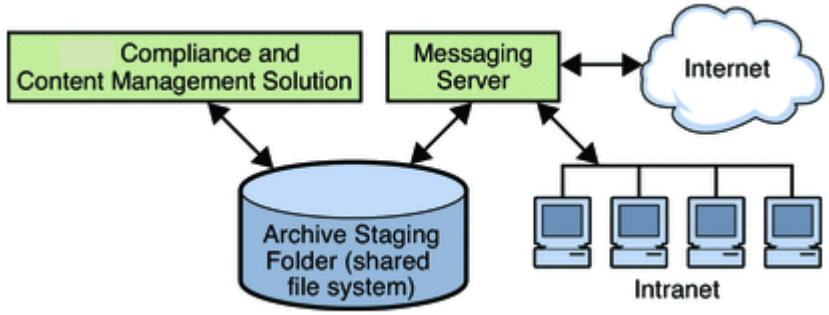
- To reduce storage usage on the message store by moving older messages to an archiving system which generally uses lower cost storage.
- As an alternative for data backup.

Note that compliance and operational archiving are not exclusive. That is, you can set up your system so that it does both compliance and operational archiving.

The Sun Compliance and Content Management Solution Theory of Operations

The interface between the AXS-One archiving system and Messaging Server consists of a shared file system called the **archive staging folder**, or simply the **staging folder**. The following figure shows a high-level architectural view.

High-level Architectural View of the Sun Compliance and Content Management Solution and Messaging Server

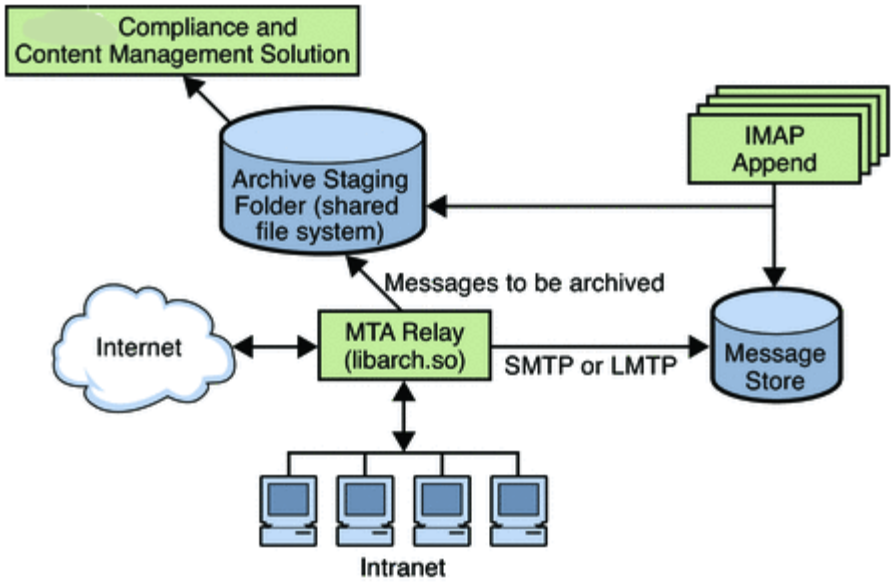


All incoming and outgoing messages are copied to the staging folder. These messages are then moved into the Sun Compliance and Content Management Solution archive system where they can be retrieved by an AXS-One client or from the Messaging Server.

Compliance Archiving Theory of Operations

The following figure shows a low-level view of a compliance architecture.

Low-Level Architectural View of the Sun Compliance and Content Management Solution /Messaging Server Compliance Archiving



As shown in the figure, messages to be archived are copied from the MTA relay to a staging folder where messages are moved into the Sun Compliance and Content Management Solution at regular intervals. Archiving can be set to occur either before or after spam and virus filtering.

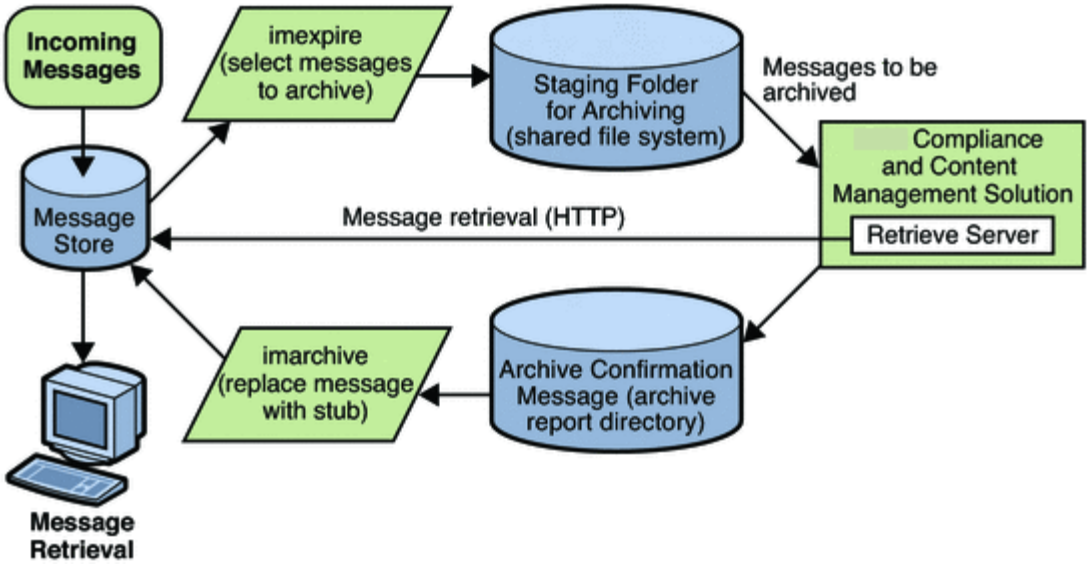
An AXS-One library file called `libarch.so` is used to implement the archiving functionality on the Messaging Server side. The **archive stream,** that is, the messages to be archived, is controlled by the Messaging Server spam filter interface. Messages can be archived on a per user, domain, channel, or per system basis (see Specifying the Messages to Be Filtered).

The arrow pointing from the IMAP Append function to the staging directory indicates messages that are moved or copied from a non-archive folder to an archive folder are archived. This is, any new message arriving into the archived part of the system is archived.

Operational Archiving Theory of Operations

In an operational archiving deployment, messages are archived from the message store instead of the MTA. The following figure shows an architectural view of an operational archiving system.

Low-Level Architectural View of the Sun Compliance and Content Management Solution/Messaging Server Operational Archiving



The previous figure shows that the `imexpire` command specifies the messages to be archived. Messages can be specified by age, size, message count, and so on (see Message Store Message Expiration). These messages are copied to the staging folder where they are archived into the Sun Compliance and Content Management Solution. The AXS-One application sends an archive confirmation message to the archive report directory indicating messages that have been successfully archived. It also provides information from which URL stubs can be constructed. `imarchive` does the following:

- Marks messages in the message store as archived and the message remains in both the message store and archive system. Marking it prevents the message from being re-archived.
- (Optional) Saves the stub of the message and deletes the RFC 822 Message.

Pre-Deployment Preparation

Before deploying an archiving system, you need to define your archiving needs and choose an appropriate architecture. See your Sun Microsystems software representative to determine the appropriate architecture.

Setting Up A Compliance Archiving Deployment

The Archive and Portal Server must be installed and configured as per the AXS-One Server documentation. Install the Archive and Portal Server on a separate host for best performance. Messaging Server should be in full operation. Note that archiving can be done with an SMTP MMP proxy located in front of the MTA host.

Compliance archiving uses both the MTA's spam filter interface and as well as the IMAP Append function to specify the message stream to be archived. To control MTA Archiving, the `option.dat` and LDAP parameters are enabled. Be sure to understand spam filter interface before setting up a compliance archiving system. Complete documentation is at Specifying the Messages to Be Filtered.

To enable IMAP Append, the `store.archive.compliance` and `store.archive.path` `configutil` options are set. If you want to set up a compliance archiving deployment that utilizes both IMAP Append as well as the MTA spam filter interface, follow the steps outlined in Configuring Messaging Server for Compliance Archiving. If you do not want to use IMAP Append to archive messages, skip **step 4** in Configuring Messaging Server for Compliance Archiving, where the `store.archive.compliance` and `store.archive.path` `configutil` options are set.

Configuring Messaging Server for Compliance Archiving

1. Make sure the Archive Server has been installed and configured as per the AXS-One documentation.
2. Make sure that the Archive and Messaging Server system users belong to the same UNIX group. The AXS-One UNIX system user name is `axsadm`. The Messaging Server UNIX system name is typically `mailsrv`. Both need to be in the same UNIX group. For more information, see the topic on creating UNIX system users and groups in *Messaging Server 8.0 Installation and Configuration Guide*.
3. Set up a shared directory for messaging archival.
   The staging directory can be a local or NFS mounted drive.
4. Ensure that the `configutil` parameters `store.archive.compliance` and `store.archive.path` are correctly set.
   `store.archive.compliance` - Set to `ON` (default).
   `store.archive.path` - Set to directory shared by the Archive Server and Messaging Server for messaging archiving in the previous step. This should be the same as the `DIRECTORY` variable in the AXS-One configuration file.
5. Edit the `option.dat` file to include following parameters:
   `SPAMFILTER`**x**`_LIBRARY=/opt/SUNWmsgsr/lib/libarch.so`
   `SPAMFILTER`**x**`_CONFIG_FILE=/opt/SUNWmsgsr/`**filename**
   `libarch.so` is the AXS-One API for Messaging. **filename** is an arbitrarily named text file containing the AXS-One configuration information (see next step).
   In both parameters, **x** is a number from 1 to 8 specifying the filtering software, in this case AXS-One. This number is used in subsequent attributes to reference the AXS-One filtering. (See step 7.)
6. Create the AXS-One configuration file. (Called **filename** in the previous step).
   This file must be readable by the `mailsrv` user. It must include the following two entries:

   ```
   STYLE=1
   DIRECTORY=Message_Archival_Staging_Area
   ```

   `STYLE` specifies the AXS-One configuration. At this time the only legal value is 1.
   `DIRECTORY` specifies the directory shared by the AXS-One server and Messaging Server for messaging archiving.

7. Specify the messages to be archived.
   Messages can be archived by user(s), domain, channel, incoming or outgoing mail. Use the spam filter interface to specify precisely the message stream you wish to archive. This is described in detail at Specifying the Messages to Be Filtered.
   Following are examples on how to specify the message stream you want to archive.
   Example 1. To archive all incoming and outgoing messages for users, specify opt-in attributes by using `LDAP_OPTIN`*x* and `LDAP_SOURCE_OPTIN`*x*, and add these attributes to your directory schema. For example:
   `LDAP_OPTIN1=AXS-One`
   `LDAP_SOURCE_OPTIN1=AXS-One`
   Now add the opt-in attribute-value pair to the LDAP user entry of any user whose mail you wish to archive. For example:
   `AXS-One: archive`
   Example 2. To archive all incoming and outgoing messages for a particular domain, specify domain opt-in attributes by using `LDAP_DOMAIN_ATTR_OPTIN`*x*. For example:
   `LDAP_OPTIN1=AXS-One`
   `LDAP_SOURCE_OPTIN1=AXS-One`
   Now add the opt-in attribute-value pair to the LDAP domain entry of the domains whose mail you wish to archive. For example:
   `AXS-One: archive`
   Archiving by Channel. Archiving by channel can provide greater flexibility and granularity for archiving, but it can also be a very complex process requiring deep knowledge of how messages flow between various channels and systems. Careful attention to the archiving requirements and to the email system and architecture is mandatory. Additionally, if the system is modified by

adding new MTAs or channels or how channels direct message flow, then the channel archiving configuration will have to be examined.
Archiving creates a tension between keeping records of everything and avoiding the creation of unnecessary and space-wasting copies. The idea is to make exactly one copy of each message as it passes through the transport infrastructure. But in order to make just one copy and not miss anything you have to understand every possible way mail can flow.
The following two examples describe archiving by channel.
Example 3. This example shows how to archive all mail sent by local users to other local users that actually gets delivered. In this example we assume that you don't care about messages inserted directly into some other user's mailbox using IMAP, or messages sent by a local user to another local user who forwards that mail to an outside address. You don't mind if some spam gets archived occasionally. You don't care what version of a given message gets archived as long as it gets saved at some point and you'd like to avoid duplication in archiving but you don't insist on it. Finally, this example assumes that ingress and egress points follow the usual norms we recommend for messaging server setups.
Such a setup would most easily be implemented by putting a `sourcespamfilterX` (or `sourcespamfilterXoptin` as required) keyword on each channel that serves as a point of ingress for local user mail and `disabledestinationspamfilterX` on each point of egress that heads out to the open Internet. (An ingress channel is the channel at which a message enters the MTA, and an egress channel is a channel at which a message leaves the MTA.)
So `sourcespamfilterX` would be put on `tcp_submit`, `tcp_auth`, and `tcp_intranet`, and `disabledestinationspamfilterX` would be put on `tcp_local`.
Typically, `tcp-local` receives inbound messages from remote SMTP hosts and outbound messages from internal users. Depending on how the system is configured, outbound messages are sent directly to remote SMTP hosts or to the smarthost/firewall system. `tcp-local` is an ingress point for incoming mail and an egress point for outgoing mail.
`disabledestinationspamfilterX` will disable spam filtering if a message came from a channel that enabled spam filtering. Thus, if a message from any of the three channels with `sourcespamfilterX` goes to `tcp-local`, it will not be archived.
Example 4. This example archives all Messaging Server incoming and outgoing messages as well as internal mail. It makes the same long list of assumptions listed previously in Example 3. Put `sourcespamfilterX` on `tcp_local`, `tcp_auth`, and `tcp_submit` on inbound relays, and put `sourcespamfilterX` on `tcp_intranet`, `tcp_auth`, and `tcp_submit` on outbound relays.

8. Compile the MTA configuration with `imsimta cnbuild` followed by `imsimta restart`.

MTA Options and Channel Keywords of Interest When Archiving

The `UNIQUE_ID_TEMPLATE` MTA option specifies a template used to convert an address into a unique identifier. The template's substitution vocabulary is the same as that for `DELIVERY_OPTIONS`. The AXS-One archiving facility will generate and use unique identifiers instead of email addresses if this option is set.

AXS-One requires the computation of a hash of each message inserted into the archive. The following MTA options control how this hash is generated:

`MESSAGE_HASH_ALGORITHM` specifies the hash algorithm. Can be any of `md2`, `md4`, `md5` (the default), `sha1`, `md128`(for RIPE-MD128), or `md160` (for RIPE-MD160).

`MESSAGE_HASH_FIELDS` -- Comma separated list of fields from the header to hash (in order). Any known header field can be specified. If this option is not specified it defaults to `message-id, from, to, cc, bcc, resent-message-id, resent-from, resent-to, resent-cc, resent-bcc, subject, content-id, content-type, content-description`.

Testing Your Compliance Archiving Deployment

1. Create two email users.
2. Set up your system as described in the previous section.

3. Send email between these users.
4. Verify if the staging directory contains `*.info`, `*.body.txt`, and `*.eml` files.

If these files exist on the staging directory, then the system is working on the messaging side.

Administering, Maintaining and Monitoring Your Compliance Archiving Deployment

- See the AXS-One documentation.

Troubleshooting Your Compliance Archiving Deployment

1. Make sure that the AXS-One and Messaging Server system users belong to the same UNIX group.
2. Verify that the spam filter interface is properly configured to capture the message stream you wish to archive.
3. Verify that the AXS-One Configuration file exists and is configured properly.
4. Check for any error messages in the `mail.log_current` file.
   Logging must be enabled (see Enabling MTA Logging).
5. Verify that the staging directory has provided write permissions to the mail server user (`mailsrv`) and its group.

Setting Up an Operational Deployment

Setting up operational archiving consists of two primary steps. In the first step you define what messages are to be archived using the `imexpire` command. The second step involves specifying what to do with the messages in the message store after archiving has occurred. That is, should the messages be replaced with a stub or not. This is specified using the `imarchive` command.

Configuring the Messaging Server

1. Make sure the Archive Server has been installed and configured as per the AXS-One Server Manual.
2. Make sure that the Archive and Messaging Server system users belong to the same UNIX group. The AXS-One UNIX system user name is `axsadm`. The Messaging Server UNIX system name is typically `mailsrv`. Both need to be in the same UNIX group. For more information, see the topic on creating UNIX system users and groups in *Messaging Server 8.0 Installation and Configuration Guide*.
3. Set up a shared directory for messaging archival.
   The staging directory can be a local or NFS mounted drive.
4. Ensure that the message store `configutil` parameters are correctly set.
   `store.archive.operational` - Enables operational archiving. Set to ON.
   `store.msghash.enable` - Enables message hash indexing. Must be set to ON for operational archiving.
   `store.archive.path` - This is the staging folder, the directory shared by the Archive Server and Messaging Server to pass messaging files for archiving. This is the directory defined in the earlier step.
   `store.archive.reportdir` - This is the directory used by the Archive Server to pass archiving reports back to Messaging Server. These reports are used by the `imarchive` command to determine when it can replace the message content on an email with a URL stub. This directory is also specified on the AXS-One side and this parameter must match that value.
   `store.archive.retrieveserver` - This is the fully qualified name of the Archive Retrieval Server that passes archived message text to Messaging Server or AXS-One client. This is the web server on which `SunJesRetrieveEML.asp` is enabled.
   `store.archive.retrievetimeout` - Specifies how many seconds to wait for the archive server to return a message before timing out. Default is 30.
5. If stubbing is required, set `MESSAGE_HASH_FIELDS` to `*` in `option.dat` file
   Example: `MESSAGE_HASH_FIELDS=*`

6. Specify the message archive policy.
   Operational archiving can archive messages by folder, user, domain, number of messages in mailbox, age of messages and so on. Use the `imexpire` interface (see Message Store Message Expiration to define the messages to be archived. The action attribute of the expire rules should be set to archive as follows:
   `action: archive`
7. Use `imarchive` to specify what to do with archived messages in the message store.
   After the AXS-One server archives an email message, it sends a confirmation message to the archive report directory with the document ID. At this point, the email message is safety archived and the email message on the message store can be replaced with a stub by running `imarchive` .
   Be sure to run `imarchive` after AXS-One has processed all the messages. If `imarchive` is run before the messages are processed, those messages could get expired again. There are a couple of ways to ensure that this won't happen. The first is to run `imarchive` before imexpire. For example:

```
# /bin/sh
imarchive -s
imexpire
```

   The other way is to simply make sure enough time has passed between running `imarchive` and `imexpire`. Use `local.schedule.`**taskname** for this (see To Schedule Automatic Tasks). In the example below, `imexpire` is run daily at 1:00am and `imarchive` is run daily at 3:00am, replacing message text with stubs:

```
configutil -o local.schedule.expire -v "0 1 * * *
/opt/SUNWmsgsr/sbin/imexpire"
configutil -o local.schedule.archive -v "0 3 * * *
/opt/SUNWmsgsr/sbin/imarchive -s"
```

Testing Your Operational Archiving Deployment

1. Set up the shared directories and the `configutil` variables as described in the previous section.
2. Create a simple test `imexpire` rule file with `archive` as the action.
   Example:

```
test.folderpattern: user/*
  test.messagesize: 1
  test.action: archive
```

3. Run `imexpired --d -f` rule_file{{ -u{}}}testuser and observe the output in `stdout`.
4. Verify if the staging directory contains `*.info`, `*.body.txt`, and `*.eml` files.
   If these files exist on the staging directory, then the system is working on the messaging side.
5. Invoke the AXS-One `extractor` command to process the files in the staging area.
6. Check the report files under the report directory.
7. Run `imarchive -v -s`.
8. Validate the stubbed messages with a mail client.

Administering, Maintaining and Monitoring Your Compliance Archiving Deployment

- See the AXS-One documentation.

Troubleshooting Your Operational Archiving Deployment

1. Make sure that the Archive and Messaging Server system users belong to the same UNIX group.
2. Verify that the `imexpire` is properly configured to capture the message stream you wish to archive.
3. Verify that the `configutil` parameters are set correctly.
4. Check for any error messages in the `default` log file.
   Logging must be enabled (see Enabling MTA Logging).
5. Verify that the staging directory and report directory have `rwx` permissions to the mail server user (example, `mailsrv`) and the AXS-One user (example, `axsadm`).
6. Verify that the `imarchive` command is configured properly.

Chapter 22. JMQ Notification

JMQ Notification

Oracle Communications Messaging Server can use GlassFish Message Queue, a standards-based messaging service, to send event notifications. GlassFish Message Queue is provided as a shared component when you install Messaging Server or other Communications Suite products.

The following page is a quick-start guide on getting JMQ notifications working along with a sample program to show the output:

- Enabling JMQ Notification (Example)

The following pages describe how to configure a JMQ notification plug-in to produce messages to be consumed by clients in a Message Queue service:

- JMQ Notification Overview
- Configuring a JMQ Notification Service (Tasks and Examples)
- JMQ Notification Messages and Properties (Reference)

Chapter 23. JMQ Notification Overview

JMQ Notification Overview

This page contains the following topics:

For related topics on JMQ notification, see the following pages:

- Enabling JMQ Notification (Example)
- Configuring a JMQ Notification Service (Tasks and Examples)
- JMQ Notification Messages and Properties (Reference)

The Messaging Server notification plug-in allows you to deliver notification messages to a messaging service or event service. The messaging service sends the notifications to consumers (client interfaces), which filter and deliver the messages to specified users.

For example, when new email arrives in a user's mailbox, the notification plug-in delivers a notification message to the messaging service. The message consumer, a component of the messaging service, receives the notification and sends it to the user's email client (such as Convergence or Mozilla Thunderbird). The email client can then display a pop-up on the user's computer screen: “You have received a new message.”

Another example: if a user's mailbox exceeds its quota, the notification plug-in produces an over-quota notification message. The message consumer sends a warning to the user and to an administrator who needs to be informed of the event.

Two Notification Messaging Services

You can configure Messaging Server to deliver notifications to two different messaging services:

- Sun Java System Message Queue 3.6 2005Q4
- Event Notification Service

The Message Queue service (JMQ) implements the Java Messaging Service (JMS) specification, providing a message broker, interfaces to create clients that produce or consume messages, and administrative services and control. JMQ follows the JMS standard for routing and delivery functions, protocols, and message formats.

The Event Notification Service is a component bundled with Messaging Server and Calendar Server. It is a proprietary service that uses a publish/subscribe architecture for sending and receiving event notifications.

You can configure a notification producer for Message Queue, for the Event Notification Service, or for both services.

> **Note**
> This chapter describes how to configure notifications for Message Queue only.

For information about the Event Notification Service, see *Unified Communications Suite Event Notification Service Guide*.

> **Note**
> ENS and JMQ event notification services are "best effort" network services. A best effort network service does not provide any guarantees that the data is delivered. More specifically, under high load situations, ENS and JMQ silently discard notifications. In addition, setting JMQ transport to "reliable" does not prevent notifications from being silently discarded. In fact, "reliable" mode may actually increase the load on the JMQ broker and thus increase the chances for dropped notifications.

Notification Plug-ins

To enable Messaging Server to produce notifications to Message Queue or the Event Notification Service, you must configure a plug-in for that service:

- The JMQ notification plug-in allows you to deliver notification messages to the Message Queue broker.
- The iBiff plug-in allows you to publish notification events to the Event Notification Service.

For information on how to load the iBiff plug-in and configure the Event Notification Service, see “Appendix B: Administering Event Notification Service in Messaging Server,” in the *Sun Java System Messaging Server Administration Guide*.

Benefits of Using JMQ Notification

The JMQ notification plug-in, with Message Queue, provides the following benefits:

- Message Queue implements the JMS standard.
- With Message Queue, you can produce messages to a *topic* or a *queue*, or to both of these delivery methods. For a brief definition, see Publishing to a Topic or a Queue.
- Message Queue offers enhanced load balancing during message distribution, especially when messages are produced to a queue.
- The JMQ notification plug-in allows you to configure up to five notification plug-ins. The different plug-ins can produce messages to a topic, to a queue, to the Event Notification Service, and so on. For details, see Using Multiple JMQ Notification Plug-ins.
- You can configure notifications conditionally for a specific set of users instead of automatically sending notifications for all users, greatly reducing the total number of messages sent.
- You can configure notifications for different sets of users to be sent to different Message Queue systems, enhancing the scalability of the notification service.
- Message Queue provides reliable delivery of notifications.
  For example, if you configure the JMQ notification plug-in to produce messages with the persistent flag enabled, the message remains in the Message Queue broker until a consumer receives it. The message is saved so that, if a server goes down, the message can be retrieved and made available to the appropriate consumer.

The following benefits are described below:

- Publishing to a Topic or a Queue
- Using Multiple JMQ Notification Plug-ins
- Configuring Parameters for a Notification Plug-in
- Configuring Conditional Notifications for Specified Users
- Sending Conditional Notifications to Distributed Message Queue Destinations

Publishing to a Topic or a Queue

A topic and queue use different messaging delivery patterns; each one can be configured in a Message Queue service.

Topic. When a message producer sends a message to a topic, a publish/subscribe architecture is used. In this broadcast pattern, a producer sends a message to a topic destination. Any number of consumers can be subscribed to this topic destination. Each consumer subscribed to the topic gets its own copy of the message. If no consumers are subscribed to the topic, the message is discarded.

The Event Notification Service also uses a publish/subscribe architecture; it is similar to the topic pattern defined in Message Queue.

Queue. When a message producer sends a message to a queue, a point-to-point architecture is used. In this pattern, a producer sends a message to a queue destination from which only one consumer can receive it. If several consumers are waiting for messages from the queue, only one subscriber will receive the message. If no consumers are waiting, the message is held until either the message times out, or a consumer expresses an interest in the queue.

Producing messages to a queue allows you to spread the message load across multiple consumers.

Using Multiple JMQ Notification Plug-ins

You can configure from one to five notification plug-ins.

Messaging Server provides a plug-in library at the following default location:

```
/opt/SUNWmsgsr/lib/libjmqnotify
```

You use the `configutil` utility to specify parameters for a plug-in and to point the plug-in to the library of executable code.

If you specify more than one plug-in, each plug-in produces notification messages independently of the others. For example, if two plug-ins are configured with a delete-message parameter, and a message is deleted from a user's mailbox, both plug-ins will produce a notification message.

By configuring multiple plug-ins, you can take advantage of different message-distribution patterns for different purposes. For example, you could configure three different plug-ins to produce messages

- To a queue (using Message Queue)
- To a topic (using Message Queue)
- To the Event Notification Service

Configuring Parameters for a Notification Plug-in

For each plug-in you configure, you must define a separate set of `configutil` parameters.

The parameters determine two kinds of information:

- The kinds of notification messages to produce. For example, enabling the `LogUser` parameter causes a notification message to be sent whenever a user logs in or out.
- Configuration information needed by Message Queue. For example, the `jmqHost` parameter identifies the IP address of the host where the Message Queue broker is running.

For instructions on how to configure a plug-in, see To Configure a JMQ Notification Plug-in.

Configuring Conditional Notifications for Specified Users

Suppose only a few users in your deployment use notifications. For example, you could have a million-mailbox deployment where 10,000 users have voice-mail notifications sent to their inboxes. Enabling notifications for all users would add unnecessary message traffic to the system.

You can configure notifications to be conditional, so that notifications are generated only for the specified users who require them. The conditional use of notifications can greatly reduce the total number of notifications sent, thus reducing the overall load on the system.

To set up conditional notifications, you do the following:

- Disable notifications for all users
- Enable notifications for those users with the LDAP attribute `mailEventNotificationDestination` added to their directory entries
- Ensure that this LDAP attribute is cached in queued mail messages

When a notification event occurs, Messaging Server looks up the user entry in the directory. If the LDAP attribute is present, a notification is sent to Message Queue. If not, no notification is generated.

For instructions on how to configure this feature, see To Enable Conditional Notifications for Specified Users.

Sending Conditional Notifications to Distributed Message Queue Destinations

You can configure notifications for different sets of users to be sent to different Message Queue destinations in a distributed Message Queue environment. For example, for one set of users, notifications can be routed to one Message Queue host; for a second set, notifications can be routed to another Message Queue host.

This feature enhances the scalability of the notification system by distributing notification messages to multiple Message Queue destinations.

It operates by extending conditional notifications, described in Configuring Conditional Notifications for Specified Users. Notifications are produced for users with the LDAP attribute `mailEventNotificationDestination` in their directory entries. You can associate this attribute with different names; depending on the name, the notification messages are sent to a particular Message Queue destination.

For instructions on how to configure this feature, see To Configure Conditional Notifications to Be Sent to Different Message Queue Destinations.

Chapter 24. JMQ Notification Messages and Properties

JMQ Notification Messages and Properties

This page contains the following topics:

For related topics on JMQ notification, see the following pages:

- Enabling JMQ Notification (Example)
- JMQ Notification Overview
- Configuring a JMQ Notification Service (Tasks and Examples)

Notification Messages

Notification messages can be generated for various kinds of events that occur in the message store. For example, when a user logs in, a `Login` message can be produced and delivered to the Message Queue broker.

A `configutil` parameter specifies each kind of message to be produced. You determine which events will generate messages by configuring various `configutil` parameters. The `configutil` parameters are referenced by one or more JMQ Notification targets.

All messages are delivered to a topic or a queue, depending on whether the destination type is set to `"topic"` or `"queue"`. For information on how to configure the Message Queue destination, see To Configure an Instance of the JMQ Notification Plug-in .

Each message is identified by the following message header:

```
MQ_MESSAGE_TYPE_HEADER_PROPERTY
```

The JMQ Notification plug-in supports the messages shown in the following table.

For a list of the `configutil` parameters that enable these messages, see Default Values of the configutil Parameters.

Table 22-1 JMQ Notification Messages

Notification Message	Description
DeleteMsg	Messages marked as "Deleted" are removed from the mailbox. This is the equivalent to IMAP expunge.
Login	User logged in from IMAP, HTTP, or POP. (This message is enabled with the configutil parameter local.store.notifyplugin.*.LogUser.enable.)
Logout	User logged out from IMAP, HTTP, or POP. (This message is enabled with the configutil parameter local.store.notifyplugin.*.LogUser.enable.)
MsgFlags	Message flags on a message have been changed. The old and new flags are carried with this message.
NewMsg	New message was received by the system into the user's mailbox. Can contain message headers and body.
OverQuota	Operation failed because the user's mailbox exceeded one of the quotas (diskquota, msgquota). The MTA channel holds the message until the quota changes or the user's mailbox count goes below the quota. If the message expires while it is being held by the MTA, it will be expunged.
PurgeMsg	Message expunged (as a result of an expired date) from the mailbox by the server process imexpire. This is a server side expunge, whereas DeleteMsg is a client side expunge. This is not a purge in the true sense of the word.
ReadMsg	Message in the mailbox was read. (In the IMAP protocol, the message was marked Seen.)
TrashMsg	Message was marked for deletion by IMAP or HTTP. The user may still see the message in the folder, depending on the mail client's configuration. The messages are to be removed from the folder when an expunge is performed.
UnderQuota	Quota went back to normal from OverQuota state.
UpdateMsg	Message was appended to the mailbox by an IMAP operation. For example, the user copied an email message to the mailbox. Can contain message headers and body.

Rules and Guidelines for Notification Messages

The following rules and guidelines apply to the supported notification messages:

- The text of most notification messages is a single blank space. (The blank space is used because Message Queue does not permit an empty message body.) The exceptions are as follows:
  - The NewMsg, UpdateMsg, and DeleteMsg messages can include a message header when configured with the maxHeaderSize parameter. You must set maxHeaderSize to a value greater than zero.
  - NewMsg and UpdateMsg message can include a message body when configured with the maxBodySize parameter. You must set maxBodySize to a value greater than zero. For NewMsg and UpdateMsg, by default the message body is not delivered (is turned off). This prevents overloading Message Queue. No other messages include a message body.
- Notification messages can be generated for changes to the INBOX alone, or to the INBOX and all other folders. The following configuration parameter allows for INBOX only (value = 0), or for both the INBOX and all other folders (value = 1):

  ```
  local.store.notifyplugin._jmqnotify_.noneInbox.enable
  ```

  The default setting is to generate messages from the INBOX only (value = 0). There is no

mechanism to select folders; all folders are included when the variable is enabled (value = 1).
- The NewMsg notification is issued only after the message is deposited in the user mailbox (as opposed to "after it was accepted by the server and queued in the message queue").
- Messages are not generated for POP3 client access.
- All messages can be suppressed by issuing XNOTNOTIFY. For example, an IMAP script used for housekeeping only (the users are not meant to be notified) might issue it to suppress all messages.

Notifications for Particular Message Types

Notifications can deliver status information about messages of different types, such as text messages, voice mail, and image data. Users often expect these heterogeneous message types to be stored in the same mail folder. For example, a user may want new text messages and voice mail to arrive in the user's cell phone inbox.

To configure these message types, you use configutil commands such as store.messagetype.enable. For information about configuring and managing message types, see Managing Message Types in the Message Store.

Once the message types have been configured, JMQ notification messages can identify the particular message types. You can write your Message Queue client to interpret notification messages by message type and deliver status information about each type to the mail client.

For example, suppose new messages of different types arrive in a user's mailbox. A NewMsg notification message can carry data to tell the user that, for example, there are seven new voice mail messages and four new text messages in the user's inbox.

The following notification messages can carry information that tracks particular message types:

```
NewMsg
UpdateMsg
ReadMsg
TrashMsg
DeleteMsg
PurgeMsg
OverQuota
UnderQuota
```

The JMQ notification function counts the number of messages currently in the mailbox, by message type. Instead of sending one count, an array specifying the count for each message type is sent with the notification message.

The message-specific count is carried in the numMsgs property and delivered with the notification message. For ReadMsg and TrashMsg notification messages, the number of messages seen (numSeen) and the number marked as deleted (numDeleted) are also counted by message type.

Note
The Event Notification Service does not support message types. Use a JMQ notification plug-in to deliver information about message types.

Setting Different Options on a per-messagetype Basis

You can set separate notifyplugin configuration options, including sending events to a different JMQ host, port, or type, on a per-messagetype basis. When doing so, do not create a per-message type

instance, as it is created automatically based on the base instance. You only need to set the options you want to override from the base instance.

For example, assume that you want to use per-messagetype settings and your `notifyplugin` configuration is the following:

```
local.store.notifyplugin =
/opt/sun/comms/messaging64/lib/libjmqnotify$msgjmqnotify
```

To set this configuration, you need to put the value in quotes to prevent the "$" from being interpreted by the shell. Be sure to double check the configuration after setting it.

The `configutil` options for that base instance would be of the form:

```
local.store.notifyplugin.msgjmqnotify.<option-name> = <option-value>
```

To override the notifyplugin handling of type 1 messages, use `configutil` options of the form:

```
local.store.notifyplugin.msgjmqnotify.1.<option-name> = <option-value>
```

Even though there is no "`msgjmqnotify.1`" instance created explicitly in the `local.store.notifyplugin option`, it is created automatically from the "`msgjmqnotify`" instance because `messagetype 1` is defined. The `local.store.notifyplugin.msgjmqnotify.1.` *option-name* settings are used to override the corresponding options from the `local.store.notifyplugin.msgjmqnotify.`*option-name* settings.

Default Values of the configutil Parameters

The notification messages and the configuration information needed by Message Queue are configured with `configutil` parameters.

Table 22-2 shows these parameters and their default values.

For complete definitions of the `configutil` parameters, see http://msg.wikidoc.info/index.php/Configutil_Reference.

Table 22-2 `configutil` Parameters and Their Default Values

configutil Parameter	Default Value
local.store.notifyplugin.*.annotatemsg.enable	0 — Disabled
local.store.notifyplugin.*.changeflag.enable	0 — Disabled
local.store.notifyplugin.*.copymsg.enable	0 — Disabled
local.store.notifyplugin.*.expungemsg.enable	0 — Disabled
local.store.notifyplugin.*.overquota.enable	1 — Enabled
local.store.notifyplugin.*.underquota.enable	1 — Enabled
local.store.notifyplugin.*.maxBodySize.enable	0 — Disabled
local.store.notifyplugin.*.maxHeaderSize.enable	0 — Disabled
local.store.notifyplugin.*.NewMsg.enable	1 — Enabled
local.store.notifyplugin.*.UpdateMsg.enable	1 — Enabled
local.store.notifyplugin.*.ReadMsg.enable	1 — Enabled
local.store.notifyplugin.*.DeleteMsg.enable	1 — Enabled
local.store.notifyplugin.*.PurgeMsg.enable	1 — Enabled
local.store.notifyplugin.*.LogUser.enable	1 — Enabled
local.store.notifyplugin.*.MsgFlags.enable	0 — Disabled
local.store.notifyplugin.*.nonInBox.enable	0 — Disabled
local.store.notifyplugin.*.jmqHost	"127.0.0.1"
local.store.notifyplugin.*.jmqPort	7676
local.store.notifyplugin.*.jmqTopic	"JES-MS"
local.store.notifyplugin.*.jmqQueue	"JES-MS"
local.store.notifyplugin.*.jmqUser	"guest"
local.store.notifyplugin.*.jmqPwd	"guest"
local.store.notifyplugin.*.destinationtype	"topic"
local.store.notifyplugin.*.Priority	4
local.store.notifyplugin.*.ttl	0 — Indicates that messages never time out.
local.store.notifyplugin.*.Persistent	1 — Enabled
local.store.notifyplugin.*.enable	1 — Turned on by default.
local.store.notifyplugin.*.ldapdestination	null — Turned off by default.
local.store.notifyplugin.*.msgtypes.enable	0 — Disabled

Notification Message Properties

Every message carries additional information defined in properties. Different properties are present for

different messages. For example, `NewMsg` indicates the IMAP `uid` of the new message.

Standard Notification Message Properties

Table 22-3 describes the standard notification message properties. These properties are present in all JMS messages.

Table 22-3 Standard Notification Message Properties

Property	Data Type	Description
`hostname`	ConstMQString	The host name of the machine that generated the message.
`pid`	MQInt32	ID of the process that generated the message.
`process`	ConstMQString	Specifies the name of the process that generated the message.
`timestamp`	MQFloat64	Specifies the number of milliseconds since the epoch (midnight GMT, January 1, 1970).

Properties Specific to Particular Notification Messages

Table 22-4 describes the properties carried with particular notification messages.

Each message includes a subset of the properties shown in the table below. For a list of the properties associated with each message, see Table 22-5.

Table 22-4 Properties Specific to Particular Notification Messages

Property	Data Type	Description
`attr`*n*	ConstMQString	Annotation attribute name.
`client`	ConstMQString	The IP address of the Message Queue client associated with the message.
`diskquota`	MQInt32	The disk space quota, in kilobytes, for the user associated with the message. The value is set to -1 to indicate no quotas.
`diskquotaused`	MQInt32	The amount of disk space used by the user associated with the message, in kilobytes.
`entry`*n*	ConstMQString	Annotation entry name.
`hdrLen`	MQInt32	The size of the message header. Note that this might not be the size of the header in the message body, because it might have been truncated.
`imapUid`	MQInt32	The IMAP uid property associated with the message.
`lastUid`	MQInt32	The last IMAP uid value used in the mailbox.

`mailboxName`	ConstMQstring	The message-store mailbox name associated with the event. The mailboxName has one of the following formats (where `uid` is the user's unique identifier):`uid` — identifies the inbox of a user in the default (primary) domain.`uid@domain` — identifies the inbox of a user in a hosted domain.`uid/mailboxname` — identifies the top-level mailbox of a user in the default domain. `uid@domain/mailboxname` — identifies the top-level mailbox of a user in a hosted domain.`uid/foldername/mailboxname` — identifies a mailbox in a folder of a user in the default domain. `uid@domain/foldername/mailboxname` — identifies a mailbox in a folder of a user in a hosted domain.
`msgflags`	ConstMQString	List of current message flags.
`msgquota`	MQInt32	The user's quota for the maximum number of messages. The value is set to -1 to indicate no quotas.
`newflags`	ConstMQString	The flags set for the user's mailbox message after they were changed by the current operation. This property is always present, together with `oldflags`, when a `MsgFlags` notification message is produced. For the syntax and values for `newflags`, see Syntax for newflags and oldflags Properties, below this table.
`numDeleted`	MQInt32	The number of messages in the mailbox marked as deleted. This number counts the messages deleted by the mailbox owner. If other users have access to the mailbox, their actions in the mailbox are not included in this count. (However, the other users' actions can trigger notifications such as `DeleteMsg`).
`numDeleted`*nn*	MQInt32	The total number of messages in the mailbox marked as deleted, specified for each message type. If message types are configured, a `numDeleted`*nn* property carries a count for each message type *nn*. The `numDeleted` property is always sent; it counts the total number of all messages marked as deleted, including all types.For example, if 20 messages are marked as deleted, 10 are of type 3, 7 are of type 16, and the rest are not of any recognized type, the following properties and counts are carried with the notification: `numDeleted=20numDeleted3=10 numDeleted16=7`
`numMsgs`	MQInt32	The total number of messages now in the mailbox.
`numMsgs`*nn*	MQInt32	The total number of messages now in the mailbox, specified for each message type. If message types are configured, a `numMsgs`*nn* property carries a count for each message type *nn*. The `numMsgs` property is always sent; it counts the total number of all messages in the mailbox, including all types.For example, if 20 messages are currently in the mailbox, 10 are of type 3, 7 are of type 16, and the rest are not of any recognized type, the following properties and counts are carried with the notification: `numMsgs=20numMsgs3=10numMsgs16=7`
`numSeen`	MQInt32	The number of messages in the mailbox marked as seen (read).This number counts the messages read by the mailbox owner. If other users have access to the mailbox, their actions in the mailbox are not included in this count. (However, the other users' actions can trigger notifications such as `ReadMsg`).

numSeen*nn*	MQInt32	The total number of messages in the mailbox marked as seen (read), specified for each message type. If message types are configured, a numSeen*nn* property carries a count for each message type *nn*. The numSeen property is always sent; it counts the total number of all messages marked as seen, including all types.For example, if 20 messages are marked as seen, 10 are of type 3, 7 are of type 16, and the rest are not of any recognized type, the following properties and counts are carried with the notification: numSeen=20numSeen3=10 numSeen16=7
numSeenDeleted	MQInt32	The number of messages in the mailbox marked as seen (read) and marked as deleted.This number counts the messages marked as read and deleted by the mailbox owner. If other users have access to the mailbox, their actions in the mailbox are not included in this count. (However, the other users' actions can trigger notifications such as ReadMsg}}and {{DeleteMsg).
numSeenDeleted*nn*	MQInt32	The total number of messages in the mailbox marked as seen (read) and marked as deleted, specified for each message type. If message types are configured, a numSeenDeleted*nn* property carries a count for each message type *nn*.The numSeenDeleted property is always sent; it counts the total number of all messages marked as seen and deleted, including all types.For example, if 20 messages are marked as seen and deleted, 10 are of type 3, 7 are of type 16, and the rest are not of any recognized type, the following properties and counts are carried with the notification: numSeenDeleted=20 numSeenDeleted3=10numSeenDeleted16=7
oldflags	ConstMQString	The flags set for the user's mailbox message before they were changed by the current operation. This property is always present, together with newflags, when a MsgFlags notification message is produced. For the syntax and values for oldflags, see Syntax for newflags and oldflags Properties, below this table.
quotaRoot	ConstMQString	This can be a user name, folder name, or message type.
size	MQInt32	The size of the message. Note that this may not be the size of message body, since the body is typically a truncated version of the message.
uidlist	ConstMQString	List of UIDs.
uidValidity	MQInt32	The IMAP uid validity property.
userid	ConstMQString	The userid associated with the message.

Note
Subscribers should allow for undocumented properties when parsing the message reference. This allows for future compatibility when new properties are added.

Syntax for newflags and oldflags Properties

The newflags and oldflags properties are 5--character strings. The string must have the following values:

- If the /answered flag is set, the first character is `"A"`. If not, it is blank (" ").
- If the /flagged flag is set, the second character is `"F"`. If not, it is blank (" ").
- If the /deleted flag is set, the third character is `"D"`. If not, it is blank (" ").
- If the /seen flag is set, the fourth character is `"S"`. If not, it is blank (" ").
- If the /draft flag is set, the fifth character is `"R"`. If not, it is blank (" ").

Properties Carried with Each Notification Message

Table 22-5 shows the properties associated with each notification message.

For example, to see which properties apply to a `TrashMsg` message, look in the column header for "ReadMsg, TrashMsg." A `TrashMsg` message can use `mailboxName`, `numMsgs`, `uidValidity`, `numSeen`, and `numDeleted` (in addition to the standard properties).

Table 22-5 Properties Carried with Each Notification Message

Property	newmsg, copy msg, update msg	read msg, trash msg	delete msg, purge msg	msg flags	login, logout	over quota, under quota	change flag	annotate msg	expunge msg	create	delete	rename
attrn	No	No	No	No	No	No	No	Yes	No	No	No	No
client	No	No	No	No	Yes	No	No	No	No	No	No	No
diskquota	No	No	No	No	No	Yes	No	No	No	No	No	No
diskquota used	No	No	No	No	No	Yes	No	No	No	No	No	No
entryn	No	No	No	No	No	No	No	Yes	No	No	No	No
fromUid Validity	Yes	No	No	No	No	No	No	No	No	Yes	No	Yes
hdrLen	Yes	No	No	Yes	No	No	No	No	No	No	No	No
hostname	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
imapUid	Yes	No	Yes	Yes	No	No	No	Yes	No	No	No	No
lastUid	No	No	Yes	No	No	No	No	No	Yes	No	No	No
mailbox Name	Yes	Yes	Yes	Yes	No	No	No	Yes	No	Yes	Yes	Yes
modseq	No	No	No	No	No	No	Yes	Yes	Yes	No	No	No
msgflags	No	No	No	No	No	Yes	Yes	No	No	No	No	No
msgquota	No	No	No	No	No	Yes	No	No	No	No	No	No
newflags	No	No	No	Yes	No	No	No	No	No	No	No	No
newName	No	No	No	No	No	No	No	No	No	No	No	Yes
num Deleted	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No
num Deletedn	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No	No
numMsgs	Yes	Yes	Yes	No	No	Yes	No	No	Yes	No	No	No
numMsgsn	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No	No
numSeen	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No
numSeenn	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No	No
numSeen Deleted	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No
numSeen Deletedn	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No	No
oldflags	No	No	No	Yes	No	No	No	No	No	No	No	No
operation	No	No	No	No	No	No	Yes	No	No	No	No	No
Owner	No	Yes	No	No	No	No	No	No	No	No	No	No
peruser_ flags	No	No	No	No	No	No	Yes	No	No	No	No	No
pid	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
process	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
quotaRoot	No	No	No	No	No	Yes	No	No	No	No	No	No
size	Yes	No	No	No	No	No	No	No	No	No	No	No
system_ flags	No	No	No	No	No	No	Yes	No	No	No	No	No
timestamp	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
uidlist	No	No	No	No	No	No	Yes	No	Yes	No	No	No
uidValidity	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	No	Yes
unchanged since	No	No	No	No	No	No	Yes	No	No	No	No	No
userid	No	Yes	No	No	Yes	Yes	No	No	No	No	No	No
ctx	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Note
The `numDeleted`*n*, `numMsgs`*n*, `numSeen`*n*, and `numSeenDeleted`*n* properties are carried with notifications only if message types are defined in the message store.

Chapter 25. Configuring a JMQ Notification Service (Task)

Configuring a JMQ Notification Service (Task)

This information describes how to configure a JMQ notification plug-in.

Topics:

For related topics on JMQ notification, see the following pages:

Planning for Your JMQ Notification Service

A JMQ notification plug-in is only one part of a Message Queue service. A Message Queue service also includes clients that consume the event messages and the Message Queue infrastructure (the broker, administration components, and so on).

The following steps outline the tasks you should perform to create a Message Queue service that supports Messaging Server:

1. Design your notification message service.
   Define the notification messages needed for your Messaging Server installation. The planning and design phases of your message-service development lifecycle lie outside the scope of this information. However, you should answer the following design questions before you configure the JMQ notification plug-in:
   - Which message events do you need to produce notifications? For a list of the available notification messages, see Notification Messages.
   - Do you intend to produce messages to a queue, a topic, or both?
   - Do you intend to use the proprietary Event Notification Service as well as the Message Queue service?
     The answers to these questions will help you decide whether to configure one instance of the notification plug-in or multiple instances, and to determine how to configure each instance.
2. Install, configure, and deploy the Message Queue product.
   For information about installing Message Queue, see the *Sun Java System Message Queue*

Installation Guide.
For information about configuring and deploying Message Queue, see the *Sun Java System Message Queue Administration Guide*.

3. Write one or more Message Queue clients that will consume the JMQ notification messages.
The clients must conform to the requirements for a Message Queue Java application programming interface (API). For information about writing Message Queue clients in Java, see the *Sun Java System Message Queue Developer's Guide for Java Clients*.
4. Configure and enable the JMQ notification plug-in for producing notification messages.
The remainder of this chapter describes how to configure the notification plug-in.
5. Configure and start the runtime Message Queue clients.
For information about deploying the runtime Message Queue clients, see the *Sun Java System Message Queue Administration Guide*.

To Configure an Instance of the JMQ Notification Plug-in

In this procedure, you first configure the message events that will produce notifications. Next, you specify the information needed by Message Queue. Finally (in step 10), you configure the instance name by specifying a parameter after the name of the plug-in library:

```
/opt/SUNWmsgsr/lib/libjmqnotify$<instance-name>
```

If you do not specify an instance name, `jmqnotify` is used by default.

Before You Begin

You should install, configure, and deploy the following products:

- Messaging Server
- Sun Java System Message Queue 3.6 SP3 2005Q4 or later

> **Note**
> Most of the `configutil` parameters you will configure in the following steps are optional. For a list of their default values, see Table 22-2.

1. Configure the notification event message parameters.
For each kind of notification event message you want to include in the instance, use the `local.store.notifyplugin.<instance>.<event>.enable configutil` option. For example, to enable notifications for new messages, enter:

```
configutil -o local.store.notifyplugin.jmqnotify.NewMsg.enable -v 1
```

where *jmqnotify* is the name of the default instance and `-v 1` enables notifications for this event. A value of `0` disables notifications for this event.
For a list of all the JMQ notification messages, see Notification Messages.
For definitions of the `configutil` parameters that enable the JMQ notification messages, see http://msg.wikidoc.info/index.php/Configutil_Reference.
A few notification messages use more than one `configutil` parameter to enable the message with additional features. For example, some messages can carry message headers in the notification text. For instructions on how to configure these messages, see Syntax for newflags and oldflags Properties.

Note
You must configure parameters separately for each instance you configure. Thus, if you configure two instances, named `jmq1` and `jmq2`, and you want to enable new-message notifications for both instances, you must set that option for both instances:

```
configutil -o local.store.notifyplugin.jmq1.NewMsg.enable
-v 1
configutil -o local.store.notifyplugin.jmq2.NewMsg.enable
-v 1
```

2. Specify the host where the Message Queue destination (broker) is running. For example, enter the following command:

```
configutil -o local.store.notifyplugin.<instance>.jmqHost -v
"127.0.0.1"
```

3. Specify the port for the Message Queue broker. For example, enter the following command:

```
configutil -o local.store.notifyplugin.<instance>.jmqPort -v "7676"
```

4. Specify the user ID and password of the Message Queue user authorized to produce messages to the service. For example, enter the following commands:

```
configutil -o local.store.notifyplugin.<instance>.jmqUser -v "guest"
configutil -o local.store.notifyplugin.<instance>.jmqPwd -v
"%$#a62t&amp;"
```

5. Specify whether the destination is a "`topic`" or a "`queue`". For example, enter the following command:

```
configutil -o local.store.notifyplugin.<instance>.DestinationType -v
"queue"
```

6. Specify the destination *topic* or *queue* name. For example, enter one of the following commands:

```
configutil -o local.store.notifyplugin.<instance>.jmqQueue -v "JES-MS"
```

or

```
configutil -o local.store.notifyplugin.<instance>.jmqTopic -v "JES-MS"
```

The `jmqQueue` and `jmqTopic` parameters are synonymous and mutually exclusive; you can only use one of these parameters in one instance.
`"JES-MS"` is an example name of the queue or topic to which messages will be sent.

7. Specify the Message Queue priority assigned to messages produced by this instance. For example, enter the following command:

```
configutil -o local.store.notifyplugin.<instance>.Priority -v 3
```

   The default value of the `Priority` is 4.

8. Specify the length of time (in milliseconds) that messages are retained by the Message Queue broker. For example, enter the following command:

```
configutil -o local.store.notifyplugin.<instance>.ttl -v 100
```

   The above example specifies that a message is retained by the Message Queue service for 100 milliseconds before being either delivered or discarded. A value of 0 means that a message is retained permanently; it does not time out.

9. Specify the message persistence. For example, enter the following command:

```
configutil -o local.store.notifyplugin.<instance>.Persistent -v 1
```

   The `-v 1` specifies that persistent messages are used in the Message Queue service. Allowed values are 1 (persistent) and 0 (non-persistent).

10. Configure the instance name. To configure a single instance with the default name, you can enter either the fully qualified name of the plug-in library or the name of the library and its instance name:

```
configutil -o local.store.notifyplugin -v
/opt/SUNWmsgsr/lib/libjmqnotify
```

    or

```
configutil -o local.store.notifyplugin -v
'/opt/SUNWmsgsr/lib/libjmqnotify$jmqnotify'
```

    where `/opt/SUNWmsgsr/lib/libjmqnotify` is the library name and `jmqnotify` is the default instance name. Use the dollar sign ($) to separate the library name from the instance name. Enclose the entire value in single quotes ('*value*'); if you do not, the shell will interpret the dollar sign.

The `configutil` parameters used by the default instance have names of the following form:

```
local.store.notifyplugin.jmqnotify.*
```

To configure a different instance name such as `jmq42`, you would enter the following command:

```
configutil -o local.store.notifyplugin -v
'/opt/SUNWmsgsr/lib/libjmqnotify$jmq42'
```

The `configutil` parameters read by the `jmq42` instance would have names like:

```
local.store.notifyplugin.jmq42.*
```

To Configure Multiple Instances

1. Configure a separate set of JMQ notification parameters for each instance you intend to create. For example, suppose you configure two instances named `jmq1` and `jmq2`. Assume you want to enable new-message notifications for both instances and purged-message notifications for the `jmq2` instance only. In this case, you would set the following configutil options:

   ```
   configutil -o local.store.notifyplugin.jmq1.NewMsg.enable -v 1
   configutil -o local.store.notifyplugin.jmq2.NewMsg.enable -v 1
   configutil -o local.store.notifyplugin.jmq2.PurgeMsg.enable -v 1
   ```

   You also must specify parameters that enable the instances to communicate with the Message Queue service.
   For step-by-step instructions for configuring the all the notification parameters, see To Configure an Instance of the JMQ Notification Plug-in.

2. Configure the instance names.
   To configure two instances named `jmq1` and `jmq2`, you would enter the following command:

   ```
   configutil -o local.store.notifyplugin -v \
   '/opt/SUNWmsgsr/lib/libjmqnotify$jmq1$/opt/SUNWmsgsr/lib/libjmqnotify$jmq2
   ```

   In this example, two instances of the plug-in library are run. Use a single dollar sign ($) to separate the library name from the instance name. Use a dollar sign ($) to separate the first plug-in$instance pair from the next. Enclose the entire value in single quotes ('*value*'); if you do not, the shell will interpret the dollar signs.
   In this example, the first instance builds its configuration from parameters with the name `jmq1`:

   ```
   local.store.notify.jmq1.*
   ```

   The second instance builds its configuration from parameters with the name `jmq2`:

```
local.store.notify.jmq2.*
```

Specifying Notification Messages that Use More Than One `configutil` Parameter

For most notification messages, you specify the message by running a single `local.store.notifyplugin` command.

However, the following notification messages are (or can be) configured with more than one `local.store.notifyplugin` command:

- `NewMsg`
- `UpdateMsg`
- `DeleteMsg`
- `MsgFlags`

The procedures that follow describe how to set up these notification messages.

To Configure New-Message and Updated-Message Notifications with Message Headers and Message Bodies

You can add the message headers and message bodies to the text of notification messages sent when there are new or updated email messages.

Including message headers and message bodies is optional; you can include both features, one feature only, or neither feature. The default is to send messages without message headers or message bodies.

1. Specify the new-message or updated-message notification:

   ```
   configutil -o local.store.notifyplugin.<instance>.NewMsg.enable -v 1
   configutil -o local.store.notifyplugin.<instance>.UpdateMsg.enable -v 1
   ```

2. Specify the `maxHeaderSize` parameter with a value greater than zero, as in the following example:

   ```
   configutil -o local.store.notifyplugin.<instance>.maxHeaderSize -v 1024
   ```

   The default value of `maxHeaderSize` is `0`, which sends no header information with the message.

3. Specify the `maxBodySize` parameter with a value greater than zero, as in the following example:

   ```
   configutil -o local.store.notifyplugin.<instance>.maxBodySize -v 1024
   ```

   The default value of `maxBodySize` is `0`, which sends no body with the message.

To Configure Deleted-Message Notifications with Message Headers

You can add the message headers to the text of notification messages sent when email messages are deleted.

Including message headers is optional. The default is to send notifications without message headers.

1. Enable notifications to be sent when email messages are deleted:

   ```
   configutil -o local.store.notifyplugin.<instance>.DeleteMsg.enable -v 1
   ```

2. Specify the `ExpungeHeaders` parameter:

   ```
   configutil -o local.store.notifyplugin.<instance>.ExpungeHeaders -v 1
   ```

   The `-v 1` enables message headers to be carried with deleted-message notifications. The default value of `ExpungeHeaders` is `0`, which prohibits deleted-message notifications from carrying header information.
   You must configure the `ExpungeHeaders` parameter to enable `DeleteMsg` messages to carry message headers.

3. Specify the `maxHeaderSize` parameter with a value greater than zero, as in the following example:

   ```
   configutil -o local.store.notifyplugin.<instance>.maxHeaderSize -v 1024
   ```

   The default value of `maxHeaderSize` is `0`, which sends no header information with the message.

Configuring Notifications for Changes in Message Status

You can configure a notification message to be sent when an email message has changed status.

Information Delivered in Message-Flag Notifications

A message-flags notification is produced whenever a status flag has changed because the email message was:

- Answered
- Flagged
- Deleted
- Seen (read)
- Draft

When a message-flags notification is sent, the notification carries the following properties:

- The flags set for the email message before its status changed
- The flags set for the email message after its status changed

This information is carried in two properties, `oldflags` and `newflags`, which are 5–character strings.

For a description of the values of these two properties, see Syntax for newflags and oldflags Properties.

Configutil Parameters Needed for Message-Flag Notifications

To enable message-flag notifications, you must configure the following `configutil` parameters:

- `local.store.notifyplugin.MsgFlags`
- `local.store.notifyplugin.*.MsgFlags.enable`

The first `MsgFlags` parameter enables the IMAP server and message store to identify and track the changing values of the status flags so that this information can be delivered in notification messages.

This parameter applies to all notification plug-ins. Therefore, you must enable the parameter if *any* notification plug-in/instance uses message-flag notifications. If no plug-in/instance uses message-flag notifications, be sure that this parameter is disabled (its default value).

The second parameter, `*.MsgFlags.enable`, allows message-flag notifications to be sent for a particular plug-in library.

> **Note**
> You must configure both parameters to enable notifications for message flags.

To Enable Notifications When Message-Status Flags Have Changed

1. Enable status flags to be tracked and status information to be carried with message-flag notifications:

   ```
   configutil -o local.store.notifyplugin.MsgFlags -v 1
   ```

2. Enable message-flag notifications to be sent by a particular instance:

   ```
   configutil -o local.store.notifyplugin.<instance>.MsgFlags.enable -v 1
   ```

To Enable Conditional Notifications for Specified Users

> The functionality documented in this section was introduced in **Messaging Server 7 Update 2**.

This task allows you to configure notifications to be sent for specific users who require notifications rather than for all users in your deployment. The conditional use of notifications can greatly reduce the total number of notifications sent, thus reducing the overall load on the system.

For information about how conditional notifications work, see Configuring Conditional Notifications for Specified Users.

Before You Begin

Follow all the steps in To Configure an Instance of the JMQ Notification Plug-in.

Take These Steps

1. Disable notifications for all users:

```
configutil -o local.store.notifyplugin.<instance>.enable -v 0
```

2. Specify the name of the LDAP attribute which, if present in the user's LDAP entry, will enable notifications for that user and specify the instance to use:

```
configutil -o local.store.notifyplugin.<instance>.ldapdestination -v \
   mailEventNotificationDestination
```

3. Add the attribute to the LDAP entries of the users who require notifications. For example:

```
mailEventNotificationDestination: jmqnotify
```

   where "`jmqnotify`" is the default instance name name.

4. Ensure that this LDAP attribute is cached in enqueued messages and carried in LMTP deliveries by adding the lines to `option.dat`.

```
LDAP_SPARE_1=mailEventNotificationDestination
SPARE_1_SEPARATOR=259
```

5. Rebuild the configuration file and restart Messaging Server.

```
./stop-msg
./imsimta cnbuild
./start-msg
```

To Configure Conditional Notifications to Be Sent to Different Message Queue Destinations

> The functionality documented in this section was introduced in **Messaging Server 7 Update 2**.

This task allows notifications for different sets of users to be sent to different Message Queue destinations in a distributed Message Queue environment. For example, for one set of users, notifications can be routed to one Message Queue host; for a second set, notifications can be routed to another Message Queue host.

This task begins with the similar steps as the preceding task, To Enable Conditional Notifications for Specified Users. It extends that task to allow for multiple Message Queue destinations.

For more information about this feature, see Sending Conditional Notifications to Distributed Message Queue Destinations.

Before You Begin

Follow **Step 1, Configure the notification message parameters**, in To Configure an Instance of the JMQ Notification Plug-in.

Take These Steps

1. Disable notifications for all users:

```
configutil -o local.store.notifyplugin.<instance>.enable -v 0
```

2. Enable notifications for those users with the notification attribute, `mailEventNotificationDestination`, set in their LDAP entries:

```
configutil -o local.store.notifyplugin.<instance>.ldapdestination -v \
   mailEventNotificationDestination
```

3. Add the attribute to the LDAP entries of the users who require notifications. For example:

```
mailEventNotificationDestination: <instance>
```

   You must assign to the Message Queue destination (instance). For example:

```
mailEventNotificationDestination: mqdestination1
```

   Here the LDAP attribute is set to the value "`mqdestination1`". This value can be any string. However, this value must match the JMQ destination name (instance) in the `configutil` parameters you set to provide configuration information needed by Message Queue. These parameters are listed in the next step.

4. Configure the following JMQ configuration parameters:

```
local.store.notifyplugin._jmq_destination_name_.jmqHost
local.store.notifyplugin._jmq_destination_name_.jmqPort
local.store.notifyplugin._jmq_destination_name_.jmqUser
local.store.notifyplugin._jmq_destination_name_.jmqPwd
local.store.notifyplugin._jmq_destination_name_.DestinationType

local.store.notifyplugin._jmq_destination_name_.jmqTopic
or
local.store.notifyplugin._jmq_destination_name_.jmqQueue

local.store.notifyplugin._jmq_destination_name_.Priority
local.store.notifyplugin._jmq_destination_name_.ttl
local.store.notifyplugin._jmq_destination_name_.Persistent
```

   For example:

```
configutil -o local.store.notifyplugin.mqdestination1.jmqHost -v
"127.0.0.1"
```

For details about how to configure these parameters, see **Steps 2 through 10** in To Configure an Instance of the JMQ Notification Plug-in.

5. For each different JMQ destination you want to create for different sets of users, repeat **Steps 3 and 4**, in this procedure.
   For example, to create two additional JMQ destinations named `mqdestination2` and `mqdestination3`:
   a. Use these strings in the users' `mailEventNotificationDestination` LDAP attributes. For example, for all users whose notifications will go to the second destination, add:

      ```
      mailEventNotificationDestination: messagequeuedestination2
      ```

      For all users whose notifications will go to the third destination, add:

      ```
      mailEventNotificationDestination: messagequeuedestination3
      ```

   b. Set each JMQ configuration parameter (such as `jmqHost`) to its specific value. For example:

      ```
      configutil -o local.store.notifyplugin.mqdestination2.jmqHost -v
      "127.0.0.2"
      ```

      ```
      configutil -o local.store.notifyplugin.mqdestination3.jmqHost -v
      "127.0.0.3"
      ```

      and so on for each parameter.

6. Restart Messaging Server.

```
./stop-msg
./start-msg
```

Chapter 26. Enabling JMQ Notification (Example)

Enabling JMQ Event Notification (Example)

This example contains the following sections:

- Event Notifications in Messaging Server Overview
- Enable Java Message Queue (JMQ)
- Configure and Enable the Messaging Server jmqnotify Plugin
- Verify the JMQ Broker
- Related Information

Details on JMQ integration can be found in the following pages:

- JMQ Notification Overview
- Configuring a JMQ Notification Service (Tasks and Examples)
- JMQ Notification Messages and Properties (Reference)

Event Notifications in Messaging Server Overview

Beginning with Messaging Server 6.3, there are two mechanisms for event notifications. One mechanism is the ENS event notification plugin. The second is the JMQ (Java Message Queue) event notification plugin.

The following is a quick-start guide on getting JMQ notifications working along with a sample program to show the output. These steps were tested on a Single-Host Oracle Solaris 10u3 x86 installation. For details about that installation, see the single host deployment example in *Unified Communications Suite Installation and Configuration Guide*.

Enable Java Message Queue (JMQ)

Modify the JMQ configuration file `/etc/imq/imqbrokerd.conf`.

```
replace:

AUTOSTART=NO

with:

AUTOSTART=YES
```

Start Java Message Queue.

```
/etc/init.d/imq start
```

Reset `admin/guest` password & add `jesuser` account.

```
cd /usr/bin
./imqusermgr update -u admin -p password
Are you sure you want to update user admin? (y/n) y
./imqusermgr update -u guest -p guest
Are you sure you want to update user guest? (y/n) y
./imqusermgr add -u jesuser -g user -p password
User repository for broker instance: imqbroker
User jesuser successfully added.
```

Configure and Enable the Messaging Server jmqnotify Plugin

Enable the appropriate messaging server settings.

```
cd /opt/SUNWmsgsr/sbin
./configutil -o local.store.notifyplugin.jmqnotify.newmsg.enable -v 1
./configutil -o local.store.notifyplugin.jmqnotify.updatemsg.enable -v 1
./configutil -o local.store.notifyplugin.jmqnotify.deletemsg.enable -v 1
./configutil -o local.store.notifyplugin.jmqnotify.maxheadersize -v 1024
./configutil -o local.store.notifyplugin.jmqnotify.jmqhost -v
"127.0.0.1"
./configutil -o local.store.notifyplugin.jmqnotify.jmqport -v "7676"
./configutil -o local.store.notifyplugin.jmqnotify.jmquser -v "jesuser"
./configutil -o local.store.notifyplugin.jmqnotify.jmqpwd -v "password"
./configutil -o local.store.notifyplugin.jmqnotify.destinationtype -v
"queue"
./configutil -o local.store.notifyplugin.jmqnotify.jmqqueue -v "jesms"
./configutil -o local.store.notifyplugin.jmqnotify.priority -v 3
./configutil -o local.store.notifyplugin.jmqnotify.ttl -v 1000
./configutil -o local.store.notifyplugin.jmqnotify.persistent -v 1
```

For Messaging Server 6.3 set the following:

```
./configutil -o local.store.notifyplugin -v
'/opt/SUNWmsgsr/lib/libjmqnotify$jmqnotify'
```

For Messaging Server 7.0 and later, set the following:

```
./configutil -o local.store.notifyplugin -v
'/opt/sun/comms/messaging/lib/libjmqnotify$jmqnotify'
```

Restart Messaging Server.

```
./stop-msg
./start-msg
```

Verify the JMQ Broker

You can verify that the JMQ software is operational at any time by running the following command:

```
cd /usr/bin
./imqcmd query bkr -u admin
Password: password
```

Related Information

Use the following links to find more information about producing a Java program to listen and process Messaging Server events:

- Java Message Service Tutorial
- Java Message Server Programming Interface
- Java Message Queue Developer's Guide for Java Clients

Chapter 27. Security and Access Control

Security and Access Control in Oracle Communications Messaging Server

Messaging Server supports security features that enable you to keep messages from being intercepted, prevent intruders from impersonating your users or administrators, and permit only specific people access to specific parts of your messaging system.

The Messaging Server security architecture is part of the security architecture of Oracle servers as a whole. It is built on industry standards and public protocols for maximum interoperability and consistency.

Topics:

About Server Security

About Server Security

Server security encompasses a broad set of topics. In most enterprises, ensuring that only authorized people have access to the servers, that passwords or identities are not compromised, that people do not misrepresent themselves as others when communicating, and that communications can be held confidential when necessary are all important requirements for a messaging system.

Perhaps because the security of server communication can be compromised in many ways, there are many approaches to enhancing it. This information focuses on setting up encryption, authentication, and access control. It discusses the following security-related Messaging Server topics:

- **User ID and password login:** Requiring users to enter their user IDs and passwords to log in to IMAP, POP, HTTP, or SMTP, and the use of SMTP password login to transmit sender authentication to message recipients.
- **Encryption and authentication:** Using the TLS and SSL protocols to encrypt communication and authenticate clients. Use at least Messaging Server 6.2 when using SSL. If you are running Messaging Server 5, 5.1, or 5.2, upgrade.
- **Administrator access control:** Using the access-control facilities to delegate access to a Messaging Server and some of its individual tasks.
- **TCP client access control:** Using filtering techniques to control which clients can connect to your server's POP, IMAP, HTTP, and authenticated SMTP services.

Not all security and access issues related to Messaging Server are treated in this chapter. Security topics that are discussed elsewhere include the following:

- **Physical security:** Without provisions for keeping server machines physically secure, software security can be meaningless.
- **Message-store access:** You can define a set of message-store administrators for the Messaging Server. These administrators can view and monitor mailboxes and can control access to them. For details, see Message Store Management.
- **End-user account configuration:** End-user account information can be primarily maintained by using the Delegated Administrator product.
- **Filtering unsolicited bulk email (UBE):** See Mail Filtering and Access Control.
- Secure Multipurpose Internet Mail Extensions (S/MIME) is described in Administering SMIME in Convergence.

There are a large number of documents that cover a variety of security topics. For additional background on the topics mentioned here and for other security-related information, review the security documentation on https://wikis.oracle.com.

About HTTP Security

Messaging Server supports user ID/password authentication, client certificate authentication, and Access Manager. There are some differences, however, in how the protocols handle network connections between client and server.

When a POP, IMAP, or SMTP client logs in to Messaging Server, a connection is made and a session is established. The connection lasts for the duration of the session; that is, from login to logout. When establishing a new connection, the client must reauthenticate to the server.

When an HTTP client logs in to Messaging Server, the server provides a unique session ID to the client. The client uses the session ID to establish multiple connections during a session. The HTTP client need not reauthenticate for each connection. The client need only reauthenticate if the session is dropped and

the client wants to establish a new session. (If an HTTP session remains idle for a specified time period, the server automatically drops the HTTP session and the client is logged out. The default time period is 2 hours.)

The following techniques are used to improve the security of HTTP sessions:

- The session IDs are bound to a specific IP address.
- Each session ID has a timeout value associated with it. If the session ID is not used for a specified time period, the session ID becomes invalid.
- The server keeps a database of all open session IDs, so a client cannot forge an ID.
- The session ID is stored in the URL, but not in any cookie files.

For information about specifying configuration parameters for improved connection performance, see Configuring POP, IMAP, and HTTP Services.

For information on Access Manager see Enabling Single Sign-On (SSO).

Configuring Administrator Access to Messaging Server

Configuring Administrator Access to Messaging Server

This information mostly pertains to Oracle LDAP Schema Version 1.

This information describes how to control the ways in which server administrators can gain access to Messaging Server. Administrative access to a given Messaging Server and to specific Messaging Server tasks occurs within the context of delegated server administration.

Delegated server administration is a feature of most Oracle servers. It refers to the capability of an administrator to provide other administrators with selective access to individual servers and server features. This information briefly summarizes delegated server tasks. For more detailed information, see *Delegated Administrator System Administrator's Guide*.

Topics:

Hierarchy of Delegated Administration

When you install the first Oracle server on your network, the installation program automatically creates a group in the LDAP user directory called the Configuration Administrators group. By default, the members of the Configuration Administrators group have unrestricted access to all hosts and servers on your network.

The Configuration Administrators group is at the top of an access hierarchy, such as the following, that you can create to implement delegated administration (if Oracle LDAP Schema Version 1 is used) for Messaging Server:

1. *Configuration administrator.* The "super user" for the network of Oracle servers. Has complete access to all resources.
2. *Server administrator.* A domain administrator might create groups to administer each type of server. For example, a Messaging Administrators group might be created to administer all Messaging Servers in an administrative domain or across the whole network. Members of that group have access to all Messaging Servers (but no other servers) in that administrative domain.
3. *Task administrator.* Finally, any of the above administrators might create a group, or designate an individual user, with restricted access to a single Messaging Server or a set of Messaging Servers. Such a task administrator is permitted to perform only specific, limited server tasks (such as starting or stopping the server only, or accessing logs of a given service).

Console provides convenient interfaces that allow an administrator to perform the following tasks:

- Grant a group or an individual access to a specific Messaging Server, as described in To Provide Access to the Server as a Whole.
- Restrict that access to specific tasks on a specific Messaging Server, as described in To Restrict Access to Specific Tasks.

To Provide Access to the Server as a Whole

This section describes to give a user or group permission to access a given instance of Messaging Server.

1. Log in to Console as an administrator with access to the Messaging Server you want to provide

access to.
2. Select that server in the Console window.
   From the Console menu, choose Object, then choose Set Access Permissions.
3. Add or edit the list of users and groups with access to the server.
   (For more complete instructions, see the chapter on delegating server administration in *Managing Servers with iPlanet Console*.)
   Once you have set up the list of individuals and groups that have access to the particular Messaging Server, you can then use ACIs, as described next, to delegate specific server tasks to specific people or groups on that list.

To Restrict Access to Specific Tasks

An administrator typically connects to a server to perform one or more administrative tasks. Common administrative tasks are listed in the Messaging Server Tasks form in Console.

By default, access to a particular Messaging Server means access to all of its tasks. However, each task in the Task form can have an attached set of access-control instructions (ACIs). The server consults those ACIs before giving a connected user (who must already be a user with access permissions to the server as a whole) access to any of the tasks. In fact, the server displays in the Tasks form only those tasks to which the user has permission.

If you have access to a Messaging Server, you can create or edit ACIs on any of the tasks (that is, on any of the tasks to which you have access), and thus restrict the access that other users or groups can have to them.

To Restrict the Task Access of a User or Group

1. Log in to the Console as an administrator with access to the Messaging Server you want to provide restricted access to.
2. Open the server and select a task in the server's Tasks form by clicking on the Task text.
3. From the Edit menu, choose Set Access Permissions, and add or edit the list of access rules to give a user or group the kind of access you want them to have.
4. Repeat the process for other tasks, as appropriate.
   (For more complete instructions, see the chapter on delegating server administration in *Managing Servers with iPlanet Console*.)
   ACIs and how to create them are described more fully in the chapter on delegating server administration in *Managing Servers with iPlanet Console.*

Configuring Authentication Mechanisms

Configuring Authentication Mechanisms in Messaging Server

An authentication mechanism is a particular method for a client to prove its identity to a server. Messaging Server supports authentication methods defined by the Simple Authentication and Security Layer (SASL) protocol and it supports certificate-based authentication. The SASL mechanisms are described in this information. For more information about certificate-based authentication, see Configuring Encryption and Certificate-Based Authentication.

Topics:

Overview

Messaging Server supports the following SASL authentication methods for password-based authentication.

- **PLAIN.** This mechanism passes the user's plaintext password over the network, where it is susceptible to eavesdropping.
  Secure Sockets Layer (SSL) can be used to alleviate the eavesdropping problem. For more information, see Configuring Encryption and Certificate-Based Authentication.
- **APOP.** A challenge/response authentication mechanism that can be used only with the POP3 protocol. Defined in RFC 1939. Should be used only by sites that have legacy usage of this mechanism.
- **CRAM-MD5.** A challenge/response authentication mechanism similar to APOP, but suitable for use with other protocols as well. Defined in RFC 2195. Should be used only by sites that have legacy usage of this mechanism.
- **LOGIN.** This is equivalent to PLAIN and exists only for compatibility with pre-standard implementations of SMTP authentication. This mechanism is only enabled for use by SMTP.

With a challenge/response authentication mechanism, the server sends a challenge string to the client. The client responds with a hash of that challenge and the user's password. If the client's response matches the server's own hash, the user is authenticated.

> **Note**
> The POP, IMAP, and SMTP services support all SASL mechanisms. The HTTP service supports only the plaintext password mechanism.

The following table shows some SASL and SASL-related `configutil` parameters. For the complete listing of `configutil` parameters, see the http://msg.wikidoc.info/index.php/Configutil_Reference.

Some SASL and SASL-related `configutil` Parameters

Parameter	Description
sasl.default.ldap.has_plain_passwords	Boolean to indicate that directory stores plaintext passwords which enables APOP and CRAM-MD5. Default: False
sasl.default.transition_criteria	No longer supported or used. See sasl.default.auto_transition.
sasl.default.auto_transition	Boolean. When set and a user provides a plain text password, the password storage format is transitioned to the default password storage method for the directory server. This can be used to migrate from plaintext passwords to APOP and CRAM-MD5. Default: False
service.imap.allowanonymouslogin	This enables the SASL ANONYMOUS mechanism for use by IMAP. Default: False
service.imap\pop\smtp\http.plaintextmincipher	If this is greather than 0, then disable use of plaintext passwords unless a security layer (SSL or TLS) is activated. This forces users to enable SSL or TLS on their client to log in, which prevents exposure of their passwords on the network. The MMP has an equivalent option "RestrictPlainPasswords". Note: service.smtp.plaintextmincipher was added in Messaging Server 7 Update 4. Note: Messaging Server 5.2 checked the value against the strength of the cipher negotiated by SSL or TLS. That feature has been eliminated to simplify this option and better reflect common-case usage. Default: 0
sasl.default.mech_list	A space-separated list of SASL mechanisms to enable. If non-empty, this overrides the sasl.default.ldap.has_plain_passwords option as well as the service.imap.allowanonymouslogin option. This option applies to all protocols (imap, pop, smtp). This option was removed in the 7 update 4 release. Default: False
sasl.default.ldap.searchfilter	This is the default search filter used to look up users when one is not specified in the ((inetDomainSearchFilter for the domain. The syntax is the same as inetDomainSearchFilter. (See *Unified Communications Suite Schema Reference*.) Default: (&(uid=%U)(objectclass=inetmailuser))
sasl.default.ldap.searchfordomain	By default, the authentication system looks up the domain in LDAP following the rules for domain lookup then looks up the user. However, if this option is set to "0" rather than the default value of "1", then the domain lookup does not happen and a search for the user (by using the sasl.default.ldap.searchfilter) occurs directly under the LDAP tree specified by local.ugldapbasedn. This is provided for compatibility with legacy single-domain schemas, but use is not recommended for new deployments as even a small company might go through a merger or name change that requires support for multiple domains.

To Configure Access to Plaintext Passwords

To work, the CRAM-MD5 or APOP authentication methods require access to the users' plaintext passwords. You need to perform the following steps:

1. Configure Directory Server to store passwords in cleartext.
2. Configure Messaging Server so that it knows Directory Server is using cleartext passwords.

To Configure Directory Server to Store Cleartext Passwords

To enable the CRAM-MD5 or APOP mechanisms, you must configure the Directory Server to store passwords in cleartext. If you are using a Directory Server prior to version 6, the following instructions apply. (For version 6 or later, refer to the latest Directory Server documentation).

1. In the Directory Server Console, open the Directory Server you want to configure.
2. Click the Configuration tab.
3. Open Data in the left pane.
4. Click Passwords in the right pane.
5. From the Password encryption drop-down list, choose "cleartext."

> **Note**
> This change only impacts users created in the future. You need to transition or reset existing users' passwords after this change.

To Configure Messaging Server for Cleartext Passwords

You can now configure Messaging Server so that it knows the Directory Server is able to retrieve cleartext passwords. This makes it safe for Messaging Server to advertise APOP and CRAM-MD5:

```
configutil -o sasl.default.ldap.has_plain_passwords -v 1
```

You can disable these challenge/response SASL mechanisms by setting the value to 0.

> **Note**
> Existing users cannot use APOP and CRAM-MD5 until their password is reset or migrated (see To Transition Users). Notice that MMP has an equivalent option: CRAMs.

To Transition Users

You can use `configutil` to specify information about transitioning users. An example would be if a user password changes or if a client attempts to authenticate with a mechanism for which they do not have a proper entry.

```
configutil -o sasl.default.auto_transition -v <value>
```

For value, you can specify one of the following:

- `no` or `0` - Don't transition passwords. This is the default.
- `yes` or `1` - Do transition passwords.

To successfully transition users, you must set up ACIs in the Directory Server that enable Messaging Server write access to the user password attribute. To do this, perform the steps in the next task.

To Transition Users

If you are using a Directory Server prior to version 6 the following instructions apply. For version 6 or later, refer to the latest Directory Server documentation :

1. In Console, open the Directory Server you want to configure.
2. Click the Directory tab.
3. Select the base suffix for the user/group tree.
4. From the Object menu, select Access Permissions.
5. Select (double click) the ACI for "Messaging Server End User Administrator Write Access Rights".
6. Click ACI Attributes.
7. Add the `userpassword` attribute to the list of existing attributes.
8. Click OK.
   For versions prior to 7 update 4, the `sasl.default.mech_list` option can be used to enable a list of SASL mechanisms. If non-empty, this overrides the `sasl.default.ldap.has_plain_passwords` option as well as the `service.imap.allowanonymouslogin` option. This option applies to all protocols (imap, pop, smtp).

Configuring Client Access to POP, IMAP, and HTTP Services

Configuring Client Access to POP, IMAP, and HTTP Services

Messaging Server provides two ways to control client access, one for non-dispatcher services and one for dispatcher services. This information describes the TCP client access control mechanism used by non-dispatcher services, like the IMAP and POP servers. The proxy servers, MMP and mshttpd, also use this mechanism. The internal ENS server uses the mechanism as well, but does not perform authentication and thus does not use LDAP attributes. See Mail Filtering and Access Control for the `PORT_ACCESS` mapping access control method used by dispatcher services, such as the LMTP server and the MTA's SMTP server.

> **Note**
> The MMP behaves differently with respect to access control than the other services. For example, the MMP "imap" service controls both IMAP and IMAP+SSL services (that is, controls both ports 143 and 993). The other Messaging Server services treat IMAP and IMAP+SSL as separate services, that is, IMAP+SSL on port 993 has its own access control that is separate from IMAP on port 143.

If you are managing messaging services for a large enterprise or an Internet service provider, be sure to also implement protection from spammers and DNS spoofers to improve the general security of your network. See Protecting Against Spammers who Compromise Messaging Server User Accounts for more information.

If controlling access by IP address is not an important issue for your enterprise, you do not have to create any filters. If minimal access control is all you need, see Mostly Allowing for instructions.

Topics:

How Client Access Filters Work

The Messaging Server access-control facility for TCP clients is an implementation of the TCP wrapper concept. A *TCP wrapper* is a program that listens at the same port as the TCP daemon it serves. It uses access filters to verify client identity, and it gives the client access to the daemon if the client passes the filtering process. The design of the Messaging Server TCP wrapper is based on the Unix `Tcpd` access-control facility (created by Wietse Venema).

As part of its processing, the Messaging Server TCP client access-control system performs (when necessary) the following analyses of the socket end-point addresses:

- Reverse DNS lookups of both end points (to perform name-based access control)
- Forward DNS lookups of both end points (to detect DNS spoofing)

The system compares this information against access-control statements called *filters* to decide whether to grant or deny access. For each service, separate sets of Allow filters and Deny filters control access. Allow filters explicitly grant access. Deny filters explicitly forbid access.

When a client requests access to a service, the access-control system compares the client's address or name information to each of that service's filters, in order, by using these criteria:

- The search stops at the first match. Because Allow filters are processed before Deny filters, Allow filters take precedence.
- Access is granted if the client information matches an Allow filter for that service.
- Access is denied if the client information matches a Deny filter for that service.
- If no match with any Allow or Deny filter occurs, access is granted, except in the case where there are Allow filters but no Deny filters, in which case lack of a match means that access is denied.

The filter syntax described here is flexible enough that you should be able to implement many different kinds of access-control policies in a simple and straightforward manner. You can use both Allow filters and Deny filters in any combination, even though you can probably implement most policies by using almost exclusively Allows or almost exclusively Denies.

The following sections describe filter syntax in detail and give usage examples. To Create Access Filters for Services gives the procedure for creating access filters.

Filter Syntax

Filter statements contain both service information and client information. The service information can include the name of the service, names of hosts, and addresses of hosts. The client information can include host names, and host addresses. Both the server and client information can include wildcard names or patterns.

The very simplest form of a filter is:

service`:` *hostSpec*

where *service* is the name of the service (such as `smtp`, `pop`, `imap`, or `http`) and *hostSpec* is the host name, IPv4 address, or wildcard name or pattern that represents the client requesting access. When a filter is processed, if the client seeking access matches *client*, access is either allowed or denied (depending on which type of filter this is) to the service specified by *service*. Here are some examples:

```
imap: roberts.newyork.siroe.com
pop: ALL
http: ALL
```

If these are Allow filters, the first one grants the host `roberts.newyork.siroe.com` access to the IMAP service, and the second and third grant all clients access to the POP and HTTP services, respectively. If they are Deny filters, they deny those clients access to those services. (For descriptions of wildcard names such as `ALL`, see Wildcard Names.)

Either the server or the client information in a filter can be somewhat more complex than this, in which case the filter has the more general form of:

serviceSpec`:` *clientSpec*

where *serviceSpec* can be either *service* or *service*@*hostSpec*, and *clientSpec* can be either *hostSpec* or *user*@*hostSpec*. *user* is the user name (or a wildcard name) associated with the client host seeking access. Here are two examples:

```
pop@mailServer1.siroe.com: ALL
imap: srashad@xyz.europe.siroe.com
```

If these are Deny filters, the first filter denies all clients access to the SMTP service on the host `mailServer1.siroe.com`. The second filter denies the user `srashad` at the host

`xyz.europe.siroe.com` access to the IMAP service. (For more information on when to use these expanded server and client specifications, see Server-Host Specification and Client User-Name Specification.

Finally, at its most general, a filter has the form:

serviceList`:` *clientList*

where *serviceList* consists of one or more *serviceSpec* entries, and *clientList* consists of one or more *clientSpec* entries. Individual entries within *serviceList* and *clientList* are separated by blanks and/or commas.

In this case, when a filter is processed, if the client seeking access matches any of the *clientSpec* entries in *clientList*, then access is either allowed or denied (depending on which type of filter this is) to all the services specified in *serviceList*. Here is an example:

```
pop, imap, http: .europe.siroe.com .newyork.siroe.com
```

If this is an Allow filter, it grants access to POP, IMAP, and HTTP services to all clients in either of the domains `europe.siroe.com` and `newyork.siroe.com`. For information on using a leading dot or other pattern to specify domains or subnet, see Wildcard Patterns.

You can also use the following syntax:

"+" or "-" *serviceList*:*$*next_rule*

+ (allow filter) means the daemon list services are being granted to the client list.

- (deny filter) means the services are being denied to the client list.

* (wildcard filter) allow all clients to used these services.

$ separates rules.

This example enables multiple services on all clients.

```
+imap,pop,http:*
```

This example shows multiple rules, but each rule is simplified to have only one service name and uses wildcards for the client list. (This is the most commonly used method of specifying access control in LDIF files.)

```
+imap:ALL$+pop:ALL$+http:ALL
```

An example of how to disallow all services for a user is:

```
-imap:*$-pop:*$-http:*
```

Wildcard Names

The following table shows the wildcard names to use that represent service names, host names or addresses, or user names:

Wildcard Names for Service Filters

Wildcard Name	Explanation
ALL, *	The universal wildcard. Matches all names.
LOCAL	Matches any local host (one whose name does not contain a dot character). However, if your installation uses only canonical names, even local host names will contain dots and thus will not match this wildcard.
UNKNOWN	Matches any host whose name or address is unknown. Use this wildcard name carefully: Host names may be unavailable due to temporary DNS server problems – in which case all filters that use UNKNOWN will match all client hosts. A network address is unavailable when the software cannot identify the type of network it is communicating with – in which case all filters that use UNKNOWN will match all client hosts on that network.
KNOWN	Matches any host whose name *and* address are known. Use this wildcard name carefully: Host names may be unavailable due to temporary DNS server problems – in which case all filters that use KNOWN will fail for all client hosts. A network address is unavailable when the software cannot identify the type of network it is communicating with – in which case all filters that use KNOWN will fail for all client hosts on that network.
DNSSPOOFER	Matches any host whose DNS name does not match its own IP address.

Wildcard Patterns

You can use the following patterns in service or client addresses:

- A string that begins with a dot character (.). A host name is matched if the last components of its name match the specified pattern. For example, the wildcard pattern `.siroe.com` matches all hosts in the domain `siroe.com`.
- A string of the form `n.n.n.n/m.m.m.m`. This wildcard pattern is interpreted as a *net/mask* pair. A host address is matched if *net* is equal to the bitwise AND of the address and *mask*. For example, the pattern `123.45.67.0/255.255.255.128` matches every address in the range `123.45.67.0` through `123.45.67.127`.
- A string of the form `n.n.n.n/p`. This wildcard pattern is interpreted as a *CIDR* where p is the routing prefix. The corresponding subnet mask, *mask*, is p one bits followed by 32-p zero bits for a total of 32 bits. A host address is matched if the bitwise AND of `n.n.n.n` and *mask* is equal to the bitwise AND of the address and *mask*. For example, the pattern `123.45.67.0/25` matches every address in the range `123.45.67.0` through `123.45.67.127`.

EXCEPT Operator

The access-control system supports a single operator. You can use the EXCEPT operator to create exceptions to matching names or patterns when you have multiple entries in either *serviceList* or *clientList*. For example, the expression:

list1 EXCEPT *list2*

means that anything that matches *list1* is matched, *unless* it also matches *list2*.

Here is an example:

```
ALL: ALL EXCEPT isserver.siroe.com
```

If this were a Deny filter, it would deny access to all services to all clients except those on the host machine `isserver.siroe.com`.

EXCEPT clauses can be nested. The expression:

```
list1 EXCEPT list2 EXCEPT list3
```

is evaluated as if it were:

```
list1 EXCEPT (list2 EXCEPT list3)
```

Server-Host Specification

You can further identify the specific service being requested in a filter by including server host name or address information in the *serviceSpec* entry. In that case the entry has the form:

```
service@hostSpec
```

You might want to use this feature when your Messaging Server host machine is set up for multiple internet addresses with different internet host names. If you are a service provider, you can use this facility to host multiple domains, with different access-control rules, on a single server instance.

Filter Examples

The examples in this section show a variety of approaches to controlling access. In studying the examples, keep in mind that Allow filters are processed before Deny filters, the search terminates when a match is found, and access is granted when no match is found at all.

The examples listed here use host and domain names rather than IP addresses. Remember that you can include address and netmask information in filters, which can improve reliability in the case of name-service failure.

Mostly Denying

In this case, access is denied by default. Only explicitly authorized hosts are permitted access.

The default policy (no access) is implemented with a single, trivial deny file:

```
ALL: ALL
```

This filter denies all service to all clients that have not been explicitly granted access by an Allow filter. The Allow filters, then, might be something like these:

```
ALL: LOCAL @netgroup1
ALL: .siroe.com EXCEPT externalserver.siroe.com
```

The first rule permits access from all hosts in the local domain (that is, all hosts with no dot in their host name) and from members of the group `netgroup1`. The second rule uses a leading-dot wildcard pattern to permit access from all hosts in the `siroe.com` domain, with the exception of the host `externalserver.siroe.com`.

Mostly Allowing

In this case, access is granted by default. Only explicitly specified hosts are denied access.

The default policy (access granted) makes Allow filters unnecessary. The unwanted clients are listed explicitly in Deny filters such as these:

```
ALL: externalserver.siroe1.com, .siroe.asia.com
ALL EXCEPT pop: contractor.siroe1.com, .siroe.com
```

The first filter denies all services to a particular host and to a specific domain. The second filter permits nothing but POP access from a particular host and from a specific domain.

Denying Access to Spoofed Domains

You can use the DNSSPOOFER wildcard name in a filter to detect host-name spoofing. When you specify DNSSPOOFER, the access-control system performs forward or reverse DNS lookups to verify that the client's presented host name matches its actual IP address. Here is an example for a Deny filter:

```
ALL: DNSSPOOFER
```

This filter denies all services to all remote hosts whose IP addresses don't match their DNS host names.

Controlling Access to Virtual Domains

If your messaging installation uses virtual domains, in which a single server instance is associated with multiple IP addresses and domain names, you can control access to each virtual domain through a combination of Allow and Deny filters. For example, you can use Allow filters like:

```
ALL@msgServer.siroe1.com: @.siroe1.com
ALL@msgServer.siroe2.com: @.siroe2.com
...
```

coupled with a Deny filter like:

```
ALL: ALL
```

Each Allow filter permits only hosts within `domain`*N* to connect to the service whose IP address corresponds to `msgServer.siroe`*N*`.com`. All other connections are denied.

Controlling IMAP Access While Permitting Access to Webmail

If you wish to allow users to access Webmail, but not access IMAP, create a filter like this:

`+imap:`*access_server_host*,*access_server_host*

This permits IMAP *only* from the access server hosts. You can set the filter at the IMAP server level using `service.imap.domainallowed`, or at the domain/user level with LDAP attributes.

To Create Access Filters for Services

You can create Allow and Deny filters for the IMAP, POP, or HTTP services. You can also create them for SMTP services, but they have little value because they only apply to authenticated SMTP sessions. See Mail Filtering and Access Control.

To Create Filters

You can also specify access and deny filters at the command line as follows:

- To create or edit access filters for services:

```
configutil -o service.<service>.domainallowed -v <filter>
```

where *service* is `pop`, `imap`, or `http` and *filter* follows the syntax rules described in Filter Syntax.

- To create or edit deny filters for services:

```
configutil -o service.<service>.domainnotallowed -v <filter>
```

where *service* is `pop`, `imap`, or `http` and *filter* follows the syntax rules described in Filter Syntax. For a variety of examples, see Filter Examples.

To Create Access Filters for HTTP Proxy Authentication

> Access filters for HTTP authentication has been removed starting in **Messaging Server 7 Update 4**.

Any store administrator can proxy authenticate to any service. (For more information about store administrators, see Specifying Administrator Access to the Message Store authenticate to the service if their client host is granted access by using a proxy authentication access filter.

Proxy authentication allows other services, such as a portal site, to authenticate users and pass the authentication credentials to the HTTP login service. For example, assume a portal site offers several services, one of which is Messenger Express web-based email. By using the HTTP proxy authentication feature, end users need only authenticate once to the portal service. They need not authenticate again to access their email. The portal site must configure a login server that acts as the interface between the client and the service. To help configure the login server for Messenger Express authentication, there is an authentication SDK for Messenger Express.

This section describes how to create allow filters to permit HTTP proxy authentication by IP address. This section does not describe how to set up your login server or how to use the Messenger Express authentication SDK. For more information about setting up your login server for Messenger Express and using the authentication SDK, contact your Oracle representative.

To Create Access Filters for HTTP Proxy Authentication

- Specify access filters for proxy authentication to the HTTP service at the command line as follows:

```
configutil -o service.<service>.proxydomainallowed -v <filter>
```

where *filter* follows the syntax rules described in Filter Syntax.

Configuring Encryption and Certificate-Based Authentication

Configuring Encryption and Certificate-Based Authentication

Topics:

Messaging Server uses the Transport Layer Security (TLS) protocol, otherwise known as the Secure Sockets Layer (SSL) protocol, for encrypted communications and for certificate-based authentication of clients and servers. Messaging Server supports SSL version 3.0 (not recommended, disabled by default) and TLS versions 1.0, 1.1 and 1.2.

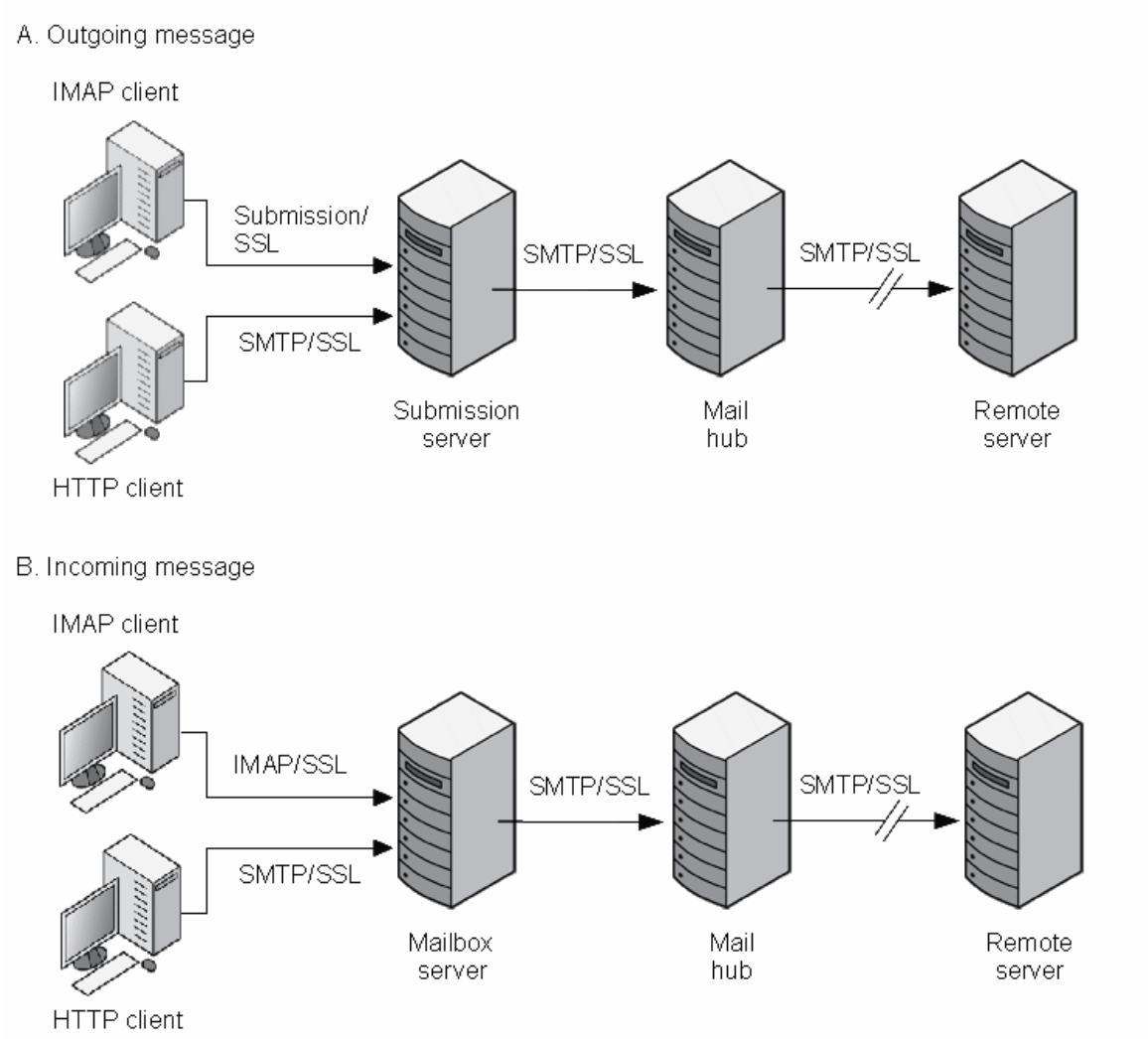
For background information on SSL, see Transport Layer Security. SSL is based on the concepts of public-key cryptography.

If transmission of messages between a Messaging Server and its clients and between the server and other servers is encrypted, there is reduced chance for eavesdropping on the communications. If connecting clients are authenticated, there is also reduced chance for intruders to impersonate (spoof) them.

SSL functions as a protocol layer beneath the application layers of IMAP4, HTTP, POP3, and SMTP. If SSL is needed with SMTP, it is normally handled on the standard SMTP relay port (25) or the standard SMTP submission port (587) with the STARTTLS command. Alternatively, SMTP submission with SSL can be handled by the port that is often used for SSL submission (465).

HTTP and HTTP/SSL require different ports. IMAP and IMAP/SSL, and POP and POP/SSL can use the same port or different ports. SSL acts at a specific stage of message communication, as shown in the following figure, for both outgoing and incoming messages.

Encrypted Communications with Messaging Server



SSL provides hop-to-hop encryption, but the message is not encrypted on each intermediate server.

> **Note**
> To enable encryption for outgoing SMTP connections, you must modify the channel definition to include the `tls` channel keywords, such as `maytls`, `musttls`, and so on. For more information, see Transport Layer Security.

There is a small performance cost to consider in setting up an SSL connection when planning server capacity. To help reduce this cost, cryptographic accelerators such as those provided by UltraSparc T1 and T2 processors can be used.

Obtaining Certificates

Whether you use SSL for encryption or for authentication, you need to obtain a server certificate for your Messaging Server. The certificate identifies your server to clients and to other servers. Another way of obtaining certificates is using `msgcert` command (described later in this section). Note that the old `certutil` command still works, but might be more complex to use and is not internationalized. For more information on `certutil`, see http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html.

This section consists of the following subsections:

- To Manage Internal and External Modules
- Creating a Password File
- Obtaining and Managing Certificates

To Manage Internal and External Modules

A server certificate establishes the ownership and validity of a key pair, the numbers used to encrypt and decrypt data. Your server's certificate and key pair represent your server's identity. They are stored in a certificate database that can be either internal to the server or on an external, removable hardware card (smartcard).

Oracle servers access a key and certificate database using a module conforming to the Public-Key Cryptography System (PKCS) #11 API. The PKCS #11 module for a given hardware device is usually obtained from its supplier and must be installed into the Messaging Server before the Messaging Server can use that device. The pre-installed "Internal PKCS # 11 Module" supports a single internal software token that uses the certificate database that is internal to the server.

Setting up the server for a certificate involves creating a database for the certificate and its keys and installing a PKCS #11 module. If you do not use an external hardware token, you create an internal database on your server, and you use the internal, default module that is part of Messaging Server. If you do use an external token, you connect a hardware smartcard reader and install its PKCS #11 module.

You can manage PKCS #11 modules, whether internal or external, through `modutil`:

1. Connect a hardware card reader to the Messaging Server host machine and install drivers.
2. To install the PKCS #11 module for the installed driver, use the `modutil` command found in `usr/sfw/bin`, which is part of the NSS/NSPR/JSS shared component installed by the Communications Suite Installer. On Red Hat Linux, you can find the `modutil` command in `usr/bin`.

> **Note**
> The Solaris operating system includes a PKCS #11 module that might maximize SPARC or x64 CPU acceleration compared to the module that is provided by NSS. More information is available at Configuring Messaging Server and UltraSPARC T1 and T2 Hardware SSL Acceleration, although this topic contains out-of-date references to 32-bit messaging servers and the deprecated `msgcert` command, which has been replaced by `certutil`.

Installing Hardware Encryption Accelerators. If you use SSL for encryption, you might be able to improve server performance in encrypting and decrypting messages by installing a hardware encryption (crypto-graphic) accelerator. An encryption accelerator typically consists of a hardware board, installed permanently in your server machine or integrated into the server CPU, plus a software driver. Messaging Server supports accelerator modules that follow the PKCS #11 API. (They are essentially hardware tokens that do not store their own keys; they use the internal database for that.) You install an accelerator by first installing the hardware and drivers as specified by the manufacturer, and then completing the installation – as with hardware certificate tokens – by installing the PKCS #11 module.

Creating a Password File

For most Oracle servers for which SSL is enabled, the administrator is prompted at startup to supply the password required to decrypt the key pair. On Messaging Server, however, to alleviate the inconvenience of having to enter the password multiple times (it is needed by at least three server processes), and to facilitate unattended server restarts, the password is read from a password file. Passwords themselves are generated when their certificate database is created using the `msgcert generate-certDB` command.

The password file is named `sslpassword.conf` and is in the directory *msg-svr-base*`/config/`. Entries in the file are individual lines with the format:

```
<moduleName>:<password>
```

where *moduleName* is the name of the (internal or external) PKCS #11 module to be used, and *password* is the password that decrypts that module's key pair. The password is stored as clear (unencrypted) text.

Messaging Server provides a default version of the password file, with the following single entry (for the internal module and default password):

```
Internal (Software) Token:password
```

If you specify anything but the default password when you install an internal certificate, you need to edit the above line of the password file to reflect the password you specified. If you install an external module, you need to add a new line to the file, containing the module name and the password you specified for it.

> **Caution**
> Because the administrator is not prompted for the module password at server startup, it is especially important that you ensure proper administrator access control to the server and proper physical security of the server host machine and its backups.

Obtaining and Managing Certificates

Whether you use SSL for encryption or for authentication, you need to obtain a server certificate for your Messaging Server. The certificate identifies your server to clients and to other servers. Starting with Messaging Server 7 Update 5, use the `certutil` command to manage certificates and key databases. For more information on the `certutil` command, see NSS Tools : certutil.

> Starting with **Messaging Server 7 Update 4**, the `msgcert` command was considered deprecated, and has been removed in **Messaging Server 7 Update 5**. The `msgcert` command's key generation and certificate request capabilities are obsolete. Use `certutil` with appropriate options (`-d` followed by a directory location with a `sql:` prefix and `-Z sha256 -g 2048`), or other third-party certificate generation tools, to create certificates and certificate requests with up-to-date security strength.

To run SSL on Messaging Server, you must either use a self-signed certificate or a Public Key Infrastructure (PKI) solution which involves an external Certificate Authority (CA). For a PKI solution, you need a CA-signed server certificate that contains both a public and a private key. This certificate is specific to one Messaging Server. You also need a trusted CA certificate, which contains a public key. The trusted CA certificate ensures that all server certificates from your CA are trusted. This certificate is sometimes also called a CA root key or root certificate. For instructions on how to create and install a self-signed CA certificate and key, see SSL/TLS Tasks.

To Enable SSL and Selecting Ciphers

SSL certificates can be installed and configured using the `msgcert` utility and by running the appropriate `configutil` commands and editing the appropriate configuration files necessary to enable SSL for the

required services.

About Ciphers

A *cipher* is the algorithm used to encrypt and decrypt data in the encryption process. SSL cipher suites control the level of protection required between SSL client and server. Different cipher suites have different properties and use different cryptographic algorithms. At any time a specific crytographic algorithm might be weakened or compromised by new research in cryptography. The ability to change the default cipher suites allows the software to adapt as security technology changes. In addition as CPUs get faster, the key size necessary to provide several years of comfortable protection increases, even if the algorithm is considered state-of-the-art.

The default set of SSL cipher suites used may change over time as more secure ones are introduced and weaker ones are deprecated. It is expected most deployments will be happy with the default set of cipher suites and it is generally not a good idea to adjust the available cipher suites without reason.

However, here are some scenarios where it may be helpful to adjust the cipher suites:

1. A site with specific security policies may wish to provide a fixed list of cipher suites to use that is set by site policy rather than simply using state-of-the-art suites provided by the NSS library. Such a site would typically configure this setting to `'-ALL,...'` where `'...'` contains the cipher suite names.
2. A site which is experimenting with higher performance or more secure cipher suites that require installation of special server certificate types, for example the elliptic curve cipher suites. Such a site would enable these additional suites once installation was complete using a setting such as `'+TLS_ECDH_RSA_WITH_AES_128_CBC_SHA'` to enable an ECDH_RSA cipher suite from RFC 4492.
3. If a site is forced to continue supporting a particularly old client that only supports weak cipher suites, they can be explicitly enabled (for example `'WEAK+DES'` enables the single-DES cipher suites.
4. In the event the cryptographic research community discovers a vulnerability in one or more of the ciphers enabled by default, this provides a mechanism to immediately disable those ciphers. For example, to disable all ciphers using the 'RC4' algorithm, simply set `'-RC4'`.

SSL Ciphers for Messaging Server

As of 29-Jan-2008, the available cipher suites in the NSS library which are used by Messaging Server to provide encryption functionality are as follows:

```
TLS_DHE_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_DSS_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_RC4_128_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA, SSL_RSA_WITH_RC4_128_MD5,
SSL_RSA_WITH_RC4_128_SHA, TLS_RSA_WITH_AES_128_CBC_SHA,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA, SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA,
SSL_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_RSA_WITH_DES_CBC_SHA, SSL_DHE_DSS_WITH_DES_CBC_SHA,
SSL_RSA_FIPS_WITH_DES_CBC_SHA, SSL_RSA_WITH_DES_CBC_SHA,
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA, TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA,
SSL_RSA_EXPORT_WITH_RC4_40_MD5,
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5, SSL_RSA_WITH_NULL_SHA,
SSL_RSA_WITH_NULL_MD5, TLS_ECDH_ECDSA_WITH_NULL_SHA,
TLS_ECDH_ECDSA_WITH_RC4_128_SHA, TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_ECDSA_WITH_NULL_SHA,
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_RSA_WITH_NULL_SHA, TLS_ECDH_RSA_WITH_RC4_128_SHA,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_RSA_WITH_NULL_SHA,
TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
```

This list excludes the SSL2 cipher suites as Messaging Server has not supported SSL2 since the Messaging Server 6.0 release. While the standard names for cipher suites (as published in TLS RFCs) are preferred, there is limited support for legacy names used in previous releases and for some OpenSSL names.

For Messaging Server 6.3 and going forward, the weak SSL cipher suites has be disabled by default. This is an incompatible change, so it's possible some old mail clients which only support export-grade SSL will break. The following configuration options can be used to turn on all cipher suites including the weak ones (but excluding the NULL ciphers):

- For MMP: default:SSLAdjustCipherSuites weak+all
- For IMAP/POP/SMTP/MSHTTPD: configutil -o local.ssladjustciphersuites -v weak+all

However, be advised to instead only turn on the specific cipher suite needed for inter-operability. For example, the common SSL_RSA_EXPORT_WITH_RC4_40_MD5 cipher suite can be enabled with: +SSL_RSA_EXPORT_WITH_RC4_40_MD5. The 56-bit ciphers are not as weak as the 40-bit ciphers so if it's possible to only enable those, the following cipher suite works: +TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA.

Adjusting the SSL Ciphers for Messaging Server

The SSL Cipher suite provided by Messaging Server for SSL communication can be adjusted using the following configuration parameter:

1. Messaging Multi-Plexor `ImapProxyAService.cfg`, `PopProxyAService.cfg`, `SmtpProxyAService.cfg`

```
default:SSLAdjustCipherSuites <cipher suite>
```

2. IMAP, POP, SMTP, and MSHTTPD (webmail) services

```
configutil -o local.ssladjustciphersuites -v <cipher suite>
```

Specify SSL Certificate

To specify the SSL certificate to be presented by Messaging Server:

```
configutil -o encryption.rsa.nssslpersonalityssl -v <certname>
```

For example:

```
# ./configutil -o encryption.rsa.nssslpersonalityssl -v "Server-Cert"
```

There is also a per-service configuration setting for the SSL server certificate nickname. The settings are as follows:

`local.imta.sslnicknames` for the SMTP and Submit servers
`local.imap.sslnicknames` for the IMAP server
`local.pop.sslnicknames` for the POP server
`local.http.sslnicknames` for web mail server

The `*.sslnicknames` options have the same meaning as (and override) the `encryption.rsa.nssslpersonalityssl` setting. Specifically, this is a comma-separated list of NSS certificate nicknames. Although more than one nickname is permitted in the list, each nickname must refer to a different type of certificate (for example, an RSA cert and a DSS cert) so the setting will almost always be only one nickname. A nickname can be unqualified in which case the NSS software token or default token will be searched, or it can have the form *"security-module*:*nickname"* in which case the specified security module will be searched for that nickname. This is needed for certificates stored in hardware tokens or places other than the default NSS database.

This does not permit the use of more than one NSS software token in the product. In particular, there is only one `cert8.db` and `key3.db` for IMAP, POP, SMTP and HTTP. NSS does not permit that.

> **Note**
> To enable SSL encryption for outgoing messages, you must modify the channel definition to include the `tls` channel keywords, such as `maytls`, `musttls`, and so on. For more information, see Transport Layer Security.

Configuring Individual Messaging Processes for SSL

This section describes the procedures you use to configure the various Messaging Server processes for SSL.

To Configure MMP for SSL

1. Set the SSL certificate `nickname`.
   Edit the `ImapProxyAService.cfg` and `PopProxyAService.cfg` files as follows, modifying

the line `default:SSLCertNickNames` to "Server-Cert".

```
# cd /opt/sun/comms/messaging64/data/config
ImapProxyAService.cfg: default:SSLCertNicknames  "Server-Cert"
PopProxyAService.cfg:  default:SSLCertNicknames  "Server-Cert"
```

2. To enable SSL and IMAP, edit the `ImapProxyAService.cfg` file and uncomment the relevant SSL settings.
   A sample `ImapProxyAService.cfg` file resembles the following:

```
# SSL configuration
#
#  Enable SSL from client to MMP with this:
default:SSLEnable        yes
default:SSLPorts         993
default:SSLCertNicknames  Server-Cert
#  Password File for SSL server keys:
default:SSLKeyPasswdFile
/opt/sun/comms/messaging64/data/config/sslpassword.conf
#  Where SSL session cache, secmod, cert, and key files are
located:
default:SSLCacheDir      /opt/sun/comms/messaging64/data/config
#  Customizable SSL security module database file name:
default:SSLSecmodFile     secmod.db
#  Customizable SSL {{cert7.db}} and {{key3.db}} file prefixes:
default:SSLCertPrefix     ""
default:SSLKeyPrefix      ""
#  Use SSL on this port when talking to the backend server (0 =
don't use SSL)
default:SSLBacksidePort   993
```

3. To enable SSL and POP, edit the `PopProxyAService.cfg` file and uncomment the relevant SSL settings.
4. Edit the `AService.cfg` file.
   For POP and SSL, add **`|995`** after the `110` in the `ServiceList` setting.
   For IMAP and SSL, add **`|993`** after `143`.
   The `AService.cfg` file should resemble the following:

```
default:ServiceList
/opt/sun/comms/messaging64/ImapProxyAService@143|993
/opt/sun/comms/messaging64/lib/PopProxyAService@110|995
```

To Configure IMAP for SSL

- Use the `configutil` command to set the following configuration parameters to enable SSL:

```
./configutil -o service.imap.enablesslport -v yes
./configutil -o service.imap.enable -v 1
./configutil -o service.imap.sslport -v 993
./configutil -o service.imap.sslusessl -v yes
```

To Configure POP for SSL

- Use the `configutil` command to set the following configuration parameters to enable SSL:

```
./configutil -o service.pop.enablesslport -v yes
./configutil -o service.pop.enable -v 1
./configutil -o service.pop.sslport -v 995
./configutil -o service.pop.sslusessl -v yes
```

To Configure HTTP for SSL

- Use the `configutil` command to set the following configuration parameters to enable SSL:

```
./configutil -o service.http.enablesslport -v yes
./configutil -o service.http.enable -v 1
./configutil -o service.http.sslport -v 443
./configutil -o service.http.sslusessl -v yes
```

To Configure SMTP for SSL

1. To enable SSL encryption for outgoing messages, modify the channel definitions to include the TLS channel keywords such as `maytls`, `musttls`, and so on.
2. Uncomment the TLS_PORT entry (in the dispatcher.cnf file located in the *Messaging-Server-Root* /config directory if you want to support SSL on an alternate port.
   For example: `port 465`

```
TLS_PORT=465
```

To Verify the SSL Configuration

1. Use the `netstat` command to verify that the service is running.
   For example:

```
netstat -an | grep <service._service_.sslport>
(where _service_ is a keyword mmp, imap, pop, http, or smtp)

e.g.

# ./configutil -o service.imap.sslport
993
# netstat -an | grep 993
      *.993                *.*                0      0 49152      0
LISTEN
```

2. Check for errors in the Messaging Server log files.
   Log files are located in the *Messaging-Server-Root*/log directories. For example, check the `imap` log to make sure that there are no SSL initialization errors (ASockSSL_Init errors).

Setting Up Certificate-Based Login

In addition to password-based authentication, Oracle servers support authentication of users through examination of their digital certificates. In certificate-based authentication, the client establishes an SSL session with the server and submits the user's certificate to the server. The server then evaluates whether the submitted certificate is genuine. If the certificate is validated, the user is considered authenticated.

To set up your Messaging Server for certificate-based login:

To Set Up Certificate-Based Login

1. Obtain a server certificate for your server. (For details, see Obtaining Certificates).
2. Install the certificates of any trusted certificate authorities that will issue certificates to the users your server will authenticate. (For details, see To Install Certificates of Trusted CAs).
   Note that as long as there is at least one trusted CA in the server's database, the server requests a client certificate from each connecting client.
3. Turn on SSL. (For details, see To Enable SSL and Selecting Ciphers)
4. (Optional) Edit your server's `certmap.conf` file so that the server appropriately searches the LDAP user directory based on information in the submitted certificates.
   Editing the `certmap.conf` file is not necessary if the email address in your users' certificates matches the email address in your users' directory entries, and you do not need to optimize searches or validate the submitted certificate against a certificate in the user entry.

Once you have taken these steps, when a client establishes an SSL session so that the user can log in to IMAP or HTTP, the Messaging Server requests the user's certificate from the client. If the certificate submitted by the client has been issued by a CA that the server has established as trusted, and if the identity in the certificate matches an entry in the user directory, the user is authenticated and access is granted (depending on access-control rules governing that user).
There is no need to disallow password-based login to enable certificate-based login.

If password-based login is allowed (which is the default state), and if you have performed the tasks described in this section, both password-based and certificate-based login are supported. In that case, if the client establishes an SSL session and supplies a certificate, certificate-based login is used. If the client does not use SSL or does not supply a certificate, the server requests a password.

Enabling POP Before SMTP

Enabling POP Before SMTP

Topics:

Enabling POP Before SMTP Overview

SMTP Authentication, or *SMTP Auth* (RFC 2554) is the preferred method of providing SMTP relay server security. SMTP Auth allows only authenticated users to send mail through the MTA. However, some legacy clients only provide support for *POP before SMTP*. If this is the case for your system, you may enable POP before SMTP as described in this information. If possible, however, encourage your users to upgrade their POP clients rather than using POP before SMTP. Once POP before SMTP is deployed at a site users become dependent on clients that fail to follow Internet security standards, putting end users at greater risk of hacking and slowing your site with the unavoidable performance penalty because of the necessity of having to track and coordinate IP addresses of recent successful POP sessions.

The Messaging Server implementation of POP before SMTP is completely different from either SIMS or Netscape Messaging Server. POP before SMTP is supported by configuring a Messaging Multiplexor (MMP) to have both a POP and SMTP proxy. When an SMTP client connects to the SMTP proxy, the proxy will check an in-memory cache of recent POP authentications. If a POP authentication from the same client IP address is found, the SMTP proxy will inform the SMTP server that it should permit messages directed to both local and non-local recipients.

To Install the SMTP Proxy

For guidelines on using the SMTP proxy see the topic on using the MMP SMTP proxy in *Unified Communications Suite Deployment Planning Guide*. To enable an MMP SMTP Proxy, set the `tcp_local_option` file (or `tcp_submit_option` file if the connection is coming in on port 587) with `PROXY_PASSWORD=<secret-string>` matching the MMP's `SmtpProxyPassword<secret-string>` value.

1. Install a Messaging Multiplexor (MMP).
   See *Unified Communications Suite Installation and Configuration Guide* for instructions.
2. Enable the SMTP proxy on the MMP.
   Add the string:
   msg-svr-base`/lib/SmtpProxyAService@25|587`
   to the `ServiceList` option in the *msg-svr-base*`/config/AService.cfg` file. That option is one long line and can't contain line breaks.

> **Note**
> When the MMP is upgraded, four new files which correspond to the existing four configuration files for the MMP are created. The new files are: `AService-def.cfg` , `ImapProxyAService-def.cfg`, `PopProxyAService-def.cfg`, and `SmtpProxyAService-def.cfg`
> The four configuration files described in the documentation are not created or affected by the install process. When the MMP starts up, it will look for the normal configuration file (as currently documented). If it doesn't find the normal configuration file, it will attempt to copy the respective `*AService-def.cfg` file to the corresponding `*AService.cfg` file name.

3. Set the PROXY_PASSWORD option in the SMTP channel option file tcp_local_option at each SMTP relay server.
   When the SMTP proxy connects to the SMTP server, it has to inform the SMTP server of the real client IP address and other connection information so that the SMTP server can correctly apply relay blocking and other security policy (including POP before SMTP authorization). This is a security sensitive operation and must be authenticated. The proxy password configured on both the MMP SMTP Proxy and the SMTP server assures that a third party cannot abuse the facility.
   Example: PROXY_PASSWORD=_A_Password_
4. Make sure the IP address that the MMP uses to connect to the SMTP server is not treated as "internal" by the INTERNAL_IP mapping table.
   See To Add SMTP Relaying for information about the INTERNAL_IP mapping table.
5. Configure the SMTP proxy to Support POP before SMTP.
6. Edit the *msg-svr-base* /config/SmtpProxyAService.cfg configuration file.
   The following SMTP proxy options operate identically to the same options for the IMAP and POP proxies (see Configuring and Administering Multiplexor Services in the Encryption (SSL) Option section.
   LdapURL, LogDir, LogLevel, BindDN, BindPass, Timeout, Banner, SSLEnable, SSLSecmodFile, SSLCertFile, SSLKeyFile, SSLKeyPasswdFile, SSLCipherSpecs, SSLCertNicknames, SSLCacheDir, SSLPorts, CertMapFile, CertmapDN, ConnLimits, TCPAccess
   Other MMP options not listed above (including the BacksidePort option) do not currently apply to the SMTP Proxy.
   Add the following five options:
   SmtpRelays is a space-separated list of SMTP relay server hostnames (with optional port) to use for round-robin relay. These relays must support the XPROXYEHLO extension. This option is mandatory with no default.
   Example: default:SmtpRelays manatee:485 gonzo mothra
   SmtpProxyPassword is a password used to authorize source channel changes on the SMTP relay servers. This option is mandatory with no default and must match the PROXY_PASSWORD option on the SMTP servers.
   Example: default:SmtpProxyPassword _A_Password_
   EhloKeywords option provides a list of EHLO extension keywords for the proxy to pass through to the client, in addition to the default set. The MMP will remove any unrecognized EHLO keywords from the EHLO list returned by an SMTP relay. EhloKeywords specifies additional EHLO keywords which should not be removed from the list. The default is empty, but the SMTP proxy will support the following keywords, so there is no need to list them in this option: 8BITMIME, PIPELINING, DSN, ENHANCEDSTATUSCODES, EXPN, HELP, XLOOP, ETRN, SIZE, STARTTLS, AUTH
   The following is an example that might be used by a site which uses the rarely used "TURN" extension:
   Example: default:EhloKeywords TURN
   PopBeforeSmtpKludgeChannel option is set to the name of an MTA channel to use for POP before SMTP authorized connections. The default is empty and the typical setting for users who want to enable POP before SMTP is tcp_intranet. This option is not required for optimizing SSL performance.
   Example: default:PopBeforeSmtpKludgeChannel tcp_intranet
   ClientLookup option defaults to no. If set to yes, a DNS reverse lookup on the client IP address will be performed unconditionally so the SMTP relay server doesn't have to do that work This option may be set on a per hosted domain basis.
   Example: default:ClientLookup yes
7. Set the PreAuth option and the AuthServiceTTL option in PopProxyAService.cfg configuration file. This option is not required for optimizing SSL performance.
   These options specify how long in seconds a user is authorized to submit mail after a POP authentication. The typical setting is 900 to 1800 (15-30 minutes).
   Example:

```
default:PreAuth yes
default:AuthServiceTTL 900
```

8. Optionally, specify how many seconds the MMP will wait for an SMTP Relay to respond before trying the next one in the list.
   The default is 10 (seconds). If a connection to an SMTP Relay fails, the MMP will avoid trying that relay for a number of minutes equivalent to the failover time-out (so if the failover time-out is 10 seconds, and a relay fails, the MMP won't try that relay again for 10 minutes).
   Example: `default:FailoverTimeout 10`

User and Group Directory Lookups Over SSL

User/Group Directory Lookups Over SSL

It is possible to do user/group directory lookups over SSL for MTA, MMP, and IMAP/POP/HTTP services. The prerequisite is that Messaging Server must be configured in SSL mode.

Set the following `configutil` parameters to enable this feature:

- `local.service.pab.ldapport` to `636`
- `local.ugldapport` to `636`
- `local.ugldapusessl` to `1`

User Password Login

User Password Login in Messaging Server

Requiring password submission on the part of users logging into Messaging Server to send or receive mail is a first line of defense against unauthorized access. Messaging Server supports password-based login for its IMAP, POP, HTTP, and SMTP services.

Topics:

IMAP, POP, and HTTP Password Login

By default, internal users must submit a password to retrieve their messages from Messaging Server. You enable or disable password login separately for POP, IMAP, and HTTP services. For more information about password login for POP, IMAP, and HTTP Services, see Password-Based Login.

User passwords can be transmitted from the user's client software to your server as cleartext or in encrypted form. If both the client and your server are configured to enable SSL and both support encryption of the required strength (as explained in To Enable SSL and Selecting Ciphers), encryption occurs.

User IDs and passwords are stored in your installation's LDAP user directory. Password security criteria, such as minimum length, are determined by directory policy requirements; they are not part of Messaging Server administration.

Certificate-based login is an alternative to password-based login. It is discussed in this chapter along with the rest of SSL; see To Set Up Certificate-Based Login

Challenge/response SASL mechanisms are another alternative to plaintext password login.

SMTP Password Login

By default, users need not submit a password when they connect to the SMTP service of Messaging Server to send a message. You can, however, enable password login to SMTP to enable authenticated SMTP.

Authenticated SMTP is an extension to the SMTP protocol that enables clients to authenticate to the server. The authentication accompanies the message. The primary use of authenticated SMTP is to allow local users who are traveling (or using their home ISP) to submit mail (relay mail) without creating an open relay that others can abuse. The "AUTH" command is used by the client to authenticate to the server.

For instructions on enabling SMTP password login (and thus Authenticated SMTP), see SMTP Authentication, SASL, and TLS.

You can use Authenticated SMTP with or without SSL encryption.

Chapter 28. Administering SMIME in Communications Express Mail

Administering S/MIME in Communications Express Mail

Support for Secure/Multipurpose Internet Mail Extension (S/MIME) 3.1 is available on Communications Express Mail. Communications Express Mail users who are set up to use S/MIME can exchange signed or encrypted messages with other users of Communications Express Mail, Microsoft Outlook Express, and Mozilla mail systems.

Information about using S/MIME in Communications Express Mail is part of the online help. Information to administer S/MIME is explained in this chapter. This chapter consists of the following sections:

Top

What is S/MIME?

Secure/Multipurpose Internet Mail Extensions (S/MIME) provides a consistent way for email users to send and receive secure MIME data, using digital signatures for authentication, message integrity and non-repudiation and encryption for privacy and data security. S/MIME version 3.1 (RFC 3851) is supported.

Several email clients support the S/MIME specification, including Microsoft Outlook Express and Mozilla mail.

You can deploy a secure mail solution by using Messaging Server and S/MIME. Communications Express webmail users who are set up to use S/MIME can exchange signed or encrypted messages with other users of Communications Express, Microsoft Outlook Express, and Mozilla mail systems. A messaging proxy can provide an additional layer of security at the firewall to further protect information assets within Messaging Server

The Communications Express webmail client supports S/MIME with these features:

- Create a digital signature for an outgoing mail message to assure the message's recipient that the message was not tampered with and is from the person who sent it
- Encrypt an outgoing mail message to prevent anyone from viewing, changing or otherwise using the message's content before the message arrives in the recipient's mailbox
- Verify the digital signature of an incoming signed message with a process involving a certificate

revocation list (CRL)
- Automatically decrypt an incoming encrypted message so the recipient can read the message's contents
- Exchange signed or encrypted messages with other users of an S/MIME compliant client such as Communications Express Mail and Mozilla mail systems

The remainder of this chapter describes how to configure Messaging Server and Communications Express for S/MIME. Note that you do not have to exclusively use Communications Express to be able to use S/MIME with Messaging Server.

Top

Concepts You Need to Know

To properly administer S/MIME, you need to be familiar with the following concepts:

- Basic administrative procedures for your platform
- Structure and use of a lightweight directory access protocol (LDAP) directory
- Addition or modification of entries in an LDAP directory
- Configuration process for the Sun Java System Directory Server
- Concepts and purpose of the following:
  - Secure Socket Layer (SSL) for a secured communications line
  - Digitally signed email messages
  - Encrypted email messages
  - Local key store of a browser
  - Smart cards and the software and hardware to use them
  - Private-public key pairs and their certificates
  - Certificate authorities (CA)
  - Verifying keys and their certificates
  - Certificate revocation list (CRL). (See 24.9.2 When is a Certificate Checked Against a CRL?)

Top

Required Software and Hardware Components

The required hardware and software for using Communications Express Mail with S/MIME is described in this section. Ensure that you install all the correct versions of the software on the server and client machines before attempting to configure for S/MIME.

Table 24-1 lists the required software and hardware for the client machine where Communications Express Mail is accessed.

Required Hardware and Software for Client Machine

Component	Description
Operating system	* Microsoft Windows 98, 2000, or XP
Browser	* Microsoft Internet Explorer, Version 6 SP2 on Windows • Microsoft Internet Explorer, Version 6 SP1 (with current patches as of December 1, 2004) on Windows 2000 and Windows 98
Sun software	Java Runtime Environment Version 1.4.2_03 or later including Java 6, but not 1.5.1, nor 1.5.4 through 1.5.12. (That is, to use Version 1.5.x, a minimum of 1.5.2 is required, but do not use 1.5.4 through 1.5.12 due to incompatibilities with Smart Cards.)
Private-public keys with certificates	One or more private-public key pair with certificates. Certificates are required and they must be in standard X.509 v3 format. Obtain keys and certificates from a CA for each Communications Express Mail user who will use the S/MIME features. The keys and their certificates are stored on the client machine or on a smart card. The public keys and certificates are also stored in an LDAP directory that can be accessed by Directory Server. A certificate revocation list (CRL), maintained by the CA, must be part of your system if you want key certificates checked against it to further ensure that the keys are valid. See 24.9.2 When is a Certificate Checked Against a CRL?
Smart card software (only required when keys and certificates are stored on smart cards)	* ActivCard Gold (now renamed ActiveIdentity), Version 2.1 or 3.0, or • NetSign, Version 3.1
Smart card reader	Any model of smart card reading device supported by the client machine and smart card software.

Table 24-2 lists the required Sun Microsystems software for the server machine.

Required Software for Server Machine

Component	Description
Mail server	Sun Java System Messaging Server 6 5 Release or later on Oracle Solaris, Version 8 or 9, and Sun SPARC machine
LDAP server	Sun Java System Directory Server 5 2004Q2
Java	Java 2 Runtime Environment, Standard Edition, Version 1.4.2 and later including Java 6, but not 1.5.1, nor 1.5.4 through 1.5.12. (That is, to use Version 1.5.x, a minimum of 1.5.2 is required, but do not use 1.5.4 through 1.5.12 due to incompatibilities with Smart Cards.)
Access Manager	(If deployment is in Schema 2) - Sun Java System Access Manager 6 2005Q1 and Communications Express - Sun Java System Communications Express 6 2005Q1 or later.

Top

Requirements for Using S/MIME

The signature and encryption features are not immediately available to Communications Express Mail users after you install Messaging Server. Before a user can take advantage of S/MIME, the requirements described in this section must be met.

Top

Private and Public Keys

At least one private and public key pair, including a certificate in standard X.509 v3 format, must be issued to each Communications Express Mail user who will use S/MIME. The certificate, used in a verification process, assures other mail users that the keys really belong to the person who uses them. A user can have more than one key pair and associated certificate.

Keys and their certificates are issued from within your organization or purchased from a third-party vendor. Regardless of how the keys and certificates are issued, the issuing organization is referred to as a certificate authority (CA).

Key pairs and their certificates are stored in two ways:

- On a smart card

  These cards are similar to commercial credit cards and should be used and safeguarded by the mail user as they do their own credit cards. Smart cards require special card readers attached to the mail user's computer (client machine) to read the private key information. See 24.3.2 Keys Stored on Smart Cards for more information.
- In a local key store on the mail user's computer (client machine)

  A mail user's browser provides the key store. The browser also provides commands to download a key pair and certificate to the key store. See 24.3.3 Keys Stored on the Client Machine for more information.

Top

Keys Stored on Smart Cards

If the private-public key pair, with its certificate, is stored on a smart card, a card reader must be properly attached to the mail user's computer. The card reading device also requires software; the device and its software are supplied by the vendor from whom you purchase this equipment.

There are actually two parts to a system with card reading capabilities. One part is the hardware card reader and it's driver. The second part is the actual card, which is usually provided by a different vendor and requires drivers for reading the cards. Not all cards are supported. Refer to the Table 24-1 to see a list of the supported SmartCards (ActiveCard, now renamed ActiveIdentity, and NetSign).

When properly installed, a mail user inserts their smart card into the reading device when they want to create a digital signature for an outgoing message. After verification of their smart card password, the private key is accessible by Communications Express Mail to sign the message. See 24.2 Required Software and Hardware Components for information on supported smart cards and reading devices.

Libraries from the vendor of the smart card are required on the user's computer. See 24.8 Key Access Libraries for the Client Machines for more information.

Top

Keys Stored on the Client Machine

If key pairs and certificates are not stored on smart cards, they must be kept in a local key store on the mail user's computer (client machine). Their browser provides the key store and also has commands to download a key pair and certificate to the key store. The key store may be password-protected; this depends on the browser.

Libraries from the vendor of the browser are required on the user's computer to support a local key store. See 24.8 Key Access Libraries for the Client Machines for more information.

Top

Publish Public Keys in LDAP Directory

All public keys and certificates must also be stored to an LDAP directory, accessible by the Sun Java System Directory Server. This is referred to as publishing the public keys so they are available to other mail users who are creating S/MIME messages.

Public keys of the sender and receiver are used in the encrypting-decrypting process of an encrypted message. Public key certificates are used to validate private keys that were used for digital signatures.

See 24.11 Managing Certificates for more information to use `ldapmodify` to publish the public keys and certificates.

Top

Give Mail Users Permission to Use S/MIME

To create a signed or encrypted message, a valid Communications Express Mail user must have permission to do so. This involves using the `mailAllowedServiceAccess` or `mailDomainAllowedServiceAccess` LDAP attributes for a user's LDAP entry. These attributes can be used to include or exclude mail users from S/MIME on an individual or domain basis.

See 24.10 Granting Permission to Use S/MIME Features for more information.

Top

Multi-language Support

A Communications Express Mail user who only uses English for their mail messages might not be able to read an S/MIME message which contains non-Latin language characters, such as Chinese. One reason for this situation is that the Java 2 Runtime Environment (JRE) installed on the user's machine does not have the `charsets.jar` file in the `/lib` directory.

The `charsets.jar` file is not installed if the English version of JRE was downloaded using the default JRE installation process. However, `charsets.jar` is installed for all other language choices of a default installation.

To ensure that the `charsets.jar` file is installed in the `/lib` directory, alert your users to use the custom installation to install the English version of JRE. During the installation process, the user must select the "Support for Additional Languages" option.

Top

Getting Started After Installing Messaging Server

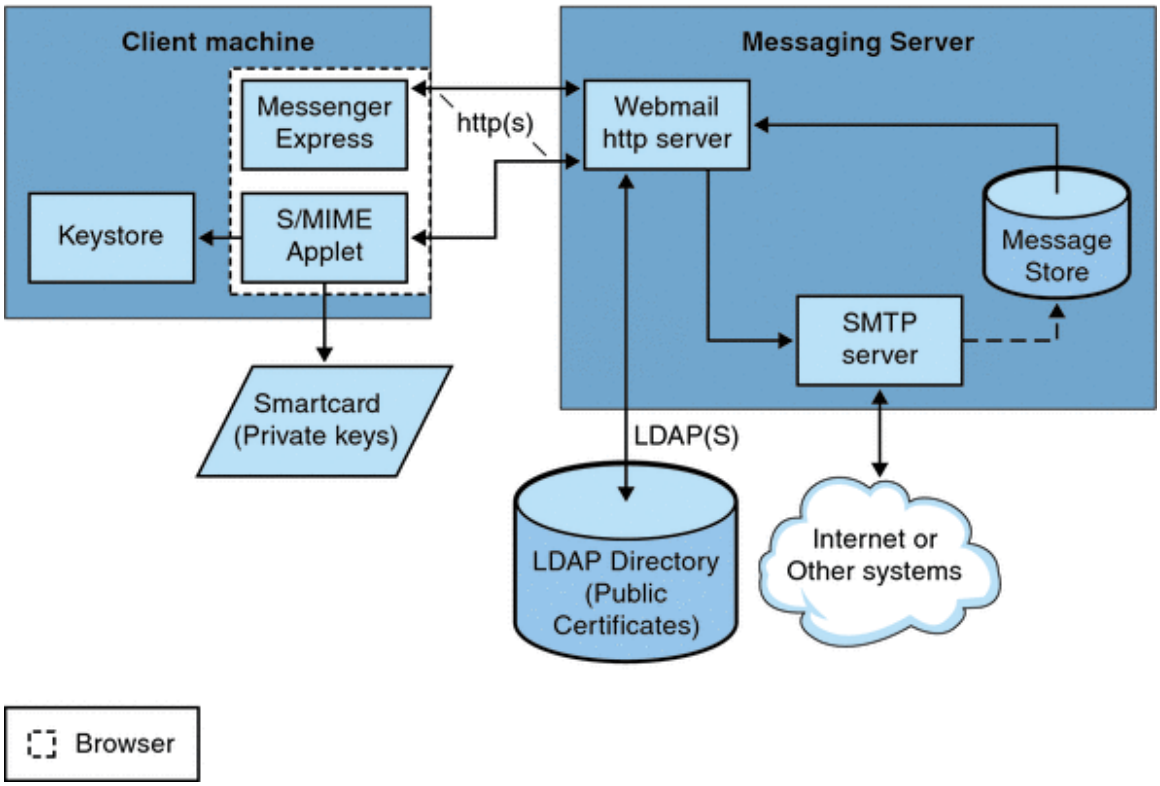
This section explains what the S/MIME applet is and provides a basic configuration procedure to set up S/MIME for Communications Express Mail. The configuration process involves setting parameters for the S/MIME applet and options for Messaging Server.

Top

The S/MIME Applet

The process of signing a message, encrypting a message, or decrypting a message, along with the various procedures to verify private and public keys, are handled by a special applet, referred to as the S/MIME applet. The configuration of the S/MIME features is done with parameters in the `smime.conf` file and options of Messaging Server. Figure 24-1 shows the S/MIME Applet in relation to other system components.

S/MIME Applet



Top

Logging In for the First Time

When a Communications Express Mail user who has permission to use S/MIME logs in to the Messaging Server for the first time, a series of special prompts displays about the S/MIME applet. After answering the prompts with Yes or Always, the S/MIME applet is downloaded to their computer. The applet remains on their machine until they log out of Communications Express Mail.

Refer to 24.11 Managing Certificates for more information.

Top

Downloading the S/MIME Applet

The S/MIME applet is downloaded each time a user logs in to Communications Express Mail unless

caching is enabled for the Java 2 Runtime Environment (JRE) on the user's machine. When caching is enabled, a copy of the S/MIME applet is saved on the user's machine after the initial download which prevents downloading the applet every time the user logs in.

Caching can improve performance so you might direct your users to do the following steps to enable caching for Java 2 Runtime Environment, Version 1.4.*x*:

To Enable Caching for Java 2 Runtime Environment, Version 1.4

1. Navigate to the Windows Control Panel.
2. Double click the Java Plug-in icon (Java 2 Runtime Environment).
3. Click the Cache tab.
4. Check the Enable Caching checkbox.
5. Click Apply.

   After downloading, a user is not aware of the S/MIME applet. It appears that signing, encrypting, or decrypting a message is done by Communications Express Mail. Unless an error message pops up, the user also is unaware of the processes to verify a private or public key. Refer to 24.9 Verifying Private and Public Keys for more information.

Top

A Basic S/MIME Configuration

The configuration file for S/MIME, `smime.conf`, contains descriptive comments and an example of each S/MIME parameter. The `smime.conf` file is included with Messaging Server, located in the directory *msg-svr-base*`/config/`, where *msg-svr-base* is the directory where Messaging Server is installed.

The following procedure contains the minimum required steps to configure the S/MIME features:

To Configure the S/MIME

1. Verify that the basic features of Communications Express Mail are working after you install Messaging Server.
2. If you haven't already, create or obtain private-public key pairs, with certificates in standard X.509 v3 format, for all your mail users who have permission to use the S/MIME features.
3. If smart cards are used for keys and certificates:
   a. Distribute the smart cards to your mail users.
   b. Ensure that the smart card reading devices and software are properly installed on each client machine where Communications Express Mail is accessed.
4. If local key stores of the browsers are used to store keys and certificates, instruct your mail users how to download their key pairs and certificate to the local key store.
5. Ensure that the correct libraries are on the client machines to support smart cards or local key stores. See 24.8 Key Access Libraries for the Client Machines
6. Set up your LDAP directory to support S/MIME:
   a. Store all certificates for the CAs in the LDAP directory, accessible by Directory Server, under the distinguished name for certificate authorities. The LDAP attribute for these certificates is `cacertificate;binary`. Write down the directory information where you store them. You'll need this information for a later step.

      See `trustedurl` in Table 24-3 for an example of specifying LDAP directory information and 24.11 Managing Certificates for information to search an LDAP directory.
   b. Store the public keys and certificates in the LDAP directory accessible by Directory Server. The LDAP attribute for public keys and certificates is `usercertificate;binary`. Write down the directory information where you store them. You'll need this information for a later step.

See `certurl` in Table 24-3 for an example of specifying LDAP directory information and 24.11 Managing Certificates for information to search an LDAP directory.

c. Ensure that all users who send or receive S/MIME messages are given permission to use S/MIME with an LDAP filter in their user entries. A filter is defined with the `mailAllowedServiceAccess` or `mailDomainAllowedServiceAccess` LDAP attributes.

> **Note**
> By default, if you do not use `mailAllowedServiceAccess` or `mailDomainAllowedServiceAccess`, all services including `smime`, are allowed. If you explicitly specify services with these attributes, then the services `http` and `smtp`, as well as `smime`, must be specified to give mail users permission to use the S/MIME features.

See 24.10 Granting Permission to Use S/MIME Features for more information.

7. Edit the `smime.conf` file with any available text editor. See comments at the beginning of the file for parameter syntax.

   All text and example parameters in `smime.conf` are preceded with a comment character (#). You can add the parameters you need to `smime.conf` or copy a parameter example to another part of the file and change its value. If you copy and edit an example, be sure to remove the # character at the beginning of its line.

   Add these parameters to the file, each on its own line:

   a. `trustedurl` (see Table 24-3)-- set to the LDAP directory information to locate the certificates of the CAs. Use the information you saved from Step a.
   b. `certurl` (Table 24-3)-- set to the LDAP directory information to locate the public keys and certificates. Use the information you saved from Step b.
   c. `usersertfilter` (see Table 24-3) – set to the value of the example in the `smime.conf` file. The example value is almost always the filter you want. Copy the example and delete the # character at the beginning of the line.

      This parameter specifies a filter definition for the primary, alternate, and equivalent email addresses of a Communications Express Mail user to ensure that all of a user's private-public key pairs are found when the key pairs are assigned to different mail addresses.
   d. `sslrootcacertsurl` (see Table 24-3)-- if you are using SSL for the communications link between the S/MIME applet and Messaging Server, set `sslrootcacertsurl` with the LDAP directory information to locate the certificates of CAs that are used to verify the Messaging Server's SSL certificates. See 24.7 Securing Internet Links With SSL for more information.

      `checkoverssl` (see Table 24-3)-- set to `0` if you are not using SSL for the communications link between the S/MIME applet and Messaging Server.
   e. `crlenable` (see Table 24-3)-- set to `0` to disable CRL checking for now because doing CRL checking might require adding other parameters to the `smime.conf` file.
   f. `logindn` and `loginpw` (Table 24-3)-- if the LDAP directory that contains the public keys and CA certificates requires authentication to access it, set these parameters to the distinguished name and password of the LDAP entry that has read permission.

> **Note**
> The values of `logindn` and `loginpw` are used whenever the LDAP directory is accessed with the LDAP information specified by the `crlmappingurl`, `sslrootcacertsurl`, or `trustedurl` parameters. See 24.5 Parameters of the smime.conf File and 24.4.3 Accessing LDAP for Public Keys, CA certificates and CRLs Using Credentials for more information.

Do not set `logindn` and `loginpw` if authentication is not required to access the LDAP directory.

8. Set the Messaging Server options with `configutil`:
   a. `local.webmail.smime.enable` – set to `1`.
   b. `local.webmail.cert.enable` – set to `1` if you want to verify certificates against a CRL.

   See 24.6 Messaging Server Options for more information.
9. Communications Express Mail is now configured for the S/MIME features. Verify that the S/MIME features are working with the following steps:
   a. Restart the Messaging Server.
   b. Check the Messaging Server log file, *msg-svr-base*`/log/http`, for diagnostic messages relating to S/MIME.
   c. If any problems were detected for S/MIME, the diagnostic messages help you determine how to correct the problem with the configuration parameters.
   d. Correct the necessary configuration parameters.
   e. Repeat Steps a. through d. until there are no more diagnostic messages for S/MIME in the Messaging Server's log file.
   f. Check that the S/MIME features are working with the following steps:
   g. Log in to Messaging Server from a client machine. Answer the special prompts for the S/MIME applet with Yes or Always. See 24.11 Managing Certificates
   h. Compose a short message, addressed to yourself.
   i. Encrypt your message by checking the Encrypt checkbox at the bottom of the Compose window if it is not already checked.
   j. Click Send to send the encrypted message to yourself. This should exercise most of the mechanisms for keys and certificates.
   k. If you find problems with the encrypted message, the most likely causes are the values you used for LDAP directory information in the `smime.conf` file and/or the way keys and certificates are stored in the LDAP directory. Check the Messaging Server log for more diagnostic messages.

   The remaining S/MIME parameters, summarized in the table below, provide many options you might want to use to further configure your S/MIME environment. See 24.5 Parameters of the smime.conf File for more information about the parameters.

Required Parameters for S/MIME	Parameters for Smart Cards and Local Key Stores	Parameters for CRL Checking	Parameters for Initial Settings and Secured Links
certurl*	platformwin	checkoverssl	alwaysencrypt
logindn		crlaccessfail	alwayssign
loginpw		crldir	sslrootcacertsurl
trustedurl*		crlenable	
usercertfilter*		crlmappingurl	
		crlurllogindn	
		crlurlloginpw	
		crlusepastnextupdate	
		readsigncert	
		revocationunknown	
		sendencryptcert	
		sendencryptcertrevoked	
		readsigncert	
		sendsigncertrevoked	
		timestampdelta	

- You must specify a value for these parameters because they have no default value.

Top

Accessing LDAP for Public Keys, CA certificates and CRLs Using Credentials

Public keys, CA certificates, and CRLs required for S/MIME may be stored in an LDAP directory (see previous section). The keys, certificates, and CRLs may be accessible from a single URL or multiple URLs in LDAP. For example, CRLs may be stored in one URL and public keys and certificates may be stored in another. Messaging Server allows you to specify which URL contains the desired CRL or certificate information, as well as the DN and password of the entry that has access to these URLs. These DN/password credentials are optional; if none are specified, LDAP access first tries the HTTP server credentials, and if that fails, it tries accessing it as `anonymous`.

Two pairs of `smime.conf` credential parameters may be set to access the desired URLs: `logindn` and `loginpw`, and `crlurllogindn` and `crlurlloginpw`.

`logindn` and `loginpw` are the credentials used for all URLs in `smime.conf`. They specify the DN and password of the LDAP entry that has read permission for the public keys, their certificates, and the CA certificates as specified by the `certurl` and `trustedurl` parameters.

`crlurllogindn` and `crlurlloginpw` specifies the DN and password of the LDAP entry that has read permission for the resulting URL from the mapping table (see 24.9.3 Accessing a CRL for more information). If these credentials are NOT accepted, LDAP access is denied and no retry with other credentials is attempted. Either both parameters must be specified, or both must be empty. These parameters do not apply to the URLs that come directly from the certificate.

Top

Setting Passwords for Specific URLs

Messaging Server allows you to specifically define the DN/ password pairs for accessing the following `smime.conf` URLs: `certUrl`, `trustedUrl`, `crlmappingUrl`, `sslrootcacertsUrl`.

The syntax is as follows:

urltype URL [|*URLDN* | *URLpassword*]

Example:

```
trustedurl==ldap://mail.siroe.com:389/cn=Directory Manager, ou=people,
o=siroe.com,o=ugroot?cacertificate?sub?(objectclass=certificationauthority) |
cn=Directory manager | boomshakalaka
```

Top

Summary of Using LDAP credentials

This section summarizes the use of LDAP credentials.

- All LDAP credentials are optional; if none are specified, LDAP access first tries the HTTP server credentials, and if that fails, tries `anonymous`.

  Two pairs of `smime.conf` parameters are used as credentials for the two sets of URLs that may be specified:

  `logindn` & `loginpw` - all URLs in `smime.conf`

  `crlurllogindn` & `crlurlloginpw` - all URLs from mapping table

  These are known as the default LDAP credential pair.
- Any URL specified in `smime.conf` or via mapping CRL URLs can have an optional local LDAP credential pair specified.
- Credentials are checked in order in which each is specified:

  1) Local LDAP credential pair - if specified, only one tried

  2) Default LDAP Credential Pair - if specified, and no Local LDAP credential pair, only one tried

  3) Server - if neither Local LDAP credential pair nor default LDAP credential pair specified, first tried

  4) `anonymous` - last tried only if server fails or none specified
- If a URL has a Local LDAP credential pair specified, it is used first; if the access fails, access is denied.
- If a URL has no Local LDAP credential pair specified, the corresponding default LDAP credential pair is used; if access fails, then access is denied.

Top

Parameters of the smime.conf File

The `smime.conf` file is included with the Messaging Server, located in the directory *msg-svr-base* `/config/`, where *msg-svr-base* is the directory where Messaging Server is installed. All text and parameter examples in the file are preceded with a comment character (#).

You can add parameters with your values to the `smime.conf` file or you can edit the parameter examples. If using an example, copy the example to another part of the file, edit the parameter's value, and remove the # character at the beginning of the line.

Edit `smime.conf` with any available text editor after you install Messaging Server. The parameters, described in Table 24-3, are not case sensitive and unless otherwise stated, are not required to be set.

S/MIME Configuration Parameters in smime.conf File

Parameter	Purpose
always encrypt	Controls the initial setting for whether all outgoing messages are automatically encrypted for all Communications Express Mail users with permission to use S/MIME. Each Communications Express Mail user can override this parameter's value for their messages by using the checkboxes described in Table 24-5. Choose one of these values: `0` - do not encrypt messages. The encryption checkboxes within Communications Express Mail are displayed as unchecked. This is the default. `1` - always encrypt messages. The encryption checkboxes within Communications Express Mail are displayed as checked. Example: `alwaysencrypt==1`
alwayssign	Controls the initial setting for whether all outgoing messages are automatically signed for all Communications Express Mail users with permission to use S/MIME. Each Communications Express Mail user can override this parameter's value for their messages by using the checkboxes described in Table 24-5. Choose one of these values: `0` - do not sign messages. The signature checkboxes within Communications Express Mail are displayed as unchecked. This is the default. `1` - always sign messages. The signature checkboxes within Communications Express Mail are displayed as checked. Example: `alwaysensign==1`

certurl	Specifies the LDAP directory information to locate the public keys and certificates of Communications Express Mail users (the LDAP attribute for public keys is `usercertificate;binary`). See 24.11 Managing Certificates for more information about certificates. This parameter must point to the highest node in the user/group of the LDAP directory information tree (DIT) that includes all users that are being served by the Messaging Server. This is particularly important for sites with more than one domain; the distinguished name must be the root distinguished name of the user/group tree instead of the subtree that contains users for a single domain. This is a required parameter that you must set. Example: `certurl==ldap://mail.siroe.com:389/ou=people, o=siroe.com,o=ugroot`
checkoverssl	Controls whether an SSL communications link is used when checking a key's certificate against a CRL. See 24.7 Securing Internet Links With SSL for more information. Choose one of these values: `0` - do not use an SSL communications link. `1` - use an SSL communications link. This is the default. A problem can occur when a proxy server is used with CRL checking in effect. See 24.9.4 Proxy Server and CRL Checking

crlaccessfail	Specifies how long to wait before the Messaging Server attempts to access a CRL after it has failed to do so after multiple attempts. This parameter has no default values.
	
	Syntax:
	
	`crlaccessfail==`*number_of_failures*`:`*time_period_for_failures*`:`
	wait_time_before_retry
	
	where:
	
	number_of_failures is the number of times that the Messaging Server can fail to access a CRL during the time interval specified by *time_period_for_failures.* The value must be greater than zero.
	
	time_period_for_failures is the number of seconds over which the Messaging Server counts the failed attempts to access a CRL. The value must be greater than zero.
	
	wait_time_before_retry is the number of seconds that the Messaging Server waits, once it detects the limit on failed attempts over the specified time interval, before trying to access the CRL again. The value must be greater than zero.
	
	Example:
	
	`crlaccessfail==10:60:300`
	
	In this example, Messaging Server fails 10 times within a minute to access the CRL. It then waits 5 minutes before attempting to access the CRL again. See 24.9.7 Trouble Accessing a CRL

crldir	Specifies the directory information where the Messaging Server downloads a CRL to disk. The default is *msg-svr-base*`/data/store/mboxlist`, where *msg-svr-base* is the directory where Messaging Server is installed. See 24.9.5 Using a Stale CRL for more information.
crlenable	Controls whether a certificate is checked against a CRL. If there is a match, the certificate is considered revoked. The values of the `send*revoked` parameters in the `smime.conf` file determine whether a key with a revoked certificate is rejected or used by Communications Express Mail. See 24.9 Verifying Private and Public Keys for more information. Choose one of these values: `0`- each certificate is not checked against a CRL. `1`- each certificate is checked against a CRL. This is the default. Ensure that the `local.webmail.cert.enable` option of the Messaging Server is set to `1`, otherwise CRL checking is not done even if `crlenable` is set to `1`.

crlmappingurl	Specifies the LDAP directory information to locate the CRL mapping definitions. This parameter is only required when you have mapping definitions. See 24.9.3 Accessing a CRL optionally add the DN and password that has access to the URL. Syntax: `crlmappingurl` *URL* [|*URL DN* | *URLpassword*] Example: `crlmappingurl==ldap://mail.siroe.com:389/cn=XYZ Messaging, ou=people, o=mail.siroe.com,o=isp?msgCRLMappingRecord?sub?( objectclass=msgCRLMappingTable) | cn=Directory Manager | pAsSwOrD`
crlurllogindn	Specifies the distinguished name of the LDAP entry that has read permission for the CRL mapping definitions (not if the entry is directly from the certificate, see "Accessing a CRL" on page 904. for more information). If values for `crllogindn` and `crlloginpw` are not specified, the Messaging Server uses the log in values for the HTTP server to gain entry to the LDAP directory. If that fails, Messaging Server attempts to access the LDAP directory anonymously. Example: `crllogindn==cn=Directory Manager`
crlurlloginpw	Specifies the password, in ASCII text, for the distinguished name of the `crllogindn` parameter. If values for `crllogindn` and `crlloginpw` are not specified, Messaging Server uses the log in values for the HTTP server to gain entry to the LDAP directory. If that fails, Messaging Server attempts to access the LDAP directory anonymously. Example: `crlloginpw==zippy`
crlusepast nextupdate	Controls whether a CRL is used when the current date is past the date specified in the CRL's next-update field. See 24.9.5 Using a Stale CRL for more information. Choose one of these values: `0` - do not use the stale CRL. `1` - use the stale CRL. This is the default.
logindn	Specifies the distinguished name of the LDAP entry that has read permission for the public keys and their certificates, and the CA certificates located in the LDAP directory specified by the `certurl` and `trustedurl` parameters. If values for `logindn` and `loginpw` are not specified, the Messaging Server uses the log in values for the HTTP server to gain entry to the LDAP directory. If that fails, Messaging Server attempts to access the LDAP directory anonymously. Example: `logindn==cn=Directory Manager`

loginpw	Specifies the password, in ASCII text, for the distinguished name of the `logindn` parameter. If values for `logindn` and `loginpw` are not specified, Messaging Server uses the log in values for the HTTP server to gain entry to the LDAP directory. If that fails, Messaging Server attempts to access the LDAP directory anonymously. Example: `loginpw==SkyKing`
platformwin	Specifies one or more library names that are necessary when using smart cards or a local key store on a Windows platform. Change this parameter only if the default value does not work for your client machines. The default is: `platformwin==CAPI:library=capibridge.dll;` See 24.8 Key Access Libraries for the Client Machines for more information.
readsigncert	Controls whether a public key's certificate is checked against a CRL to verify an S/MIME digital signature when the message is read. (A private key is used to create a digital signature for a message but it cannot be checked against a CRL, so the certificate of the public key associated with the private key is checked against the CRL.) See 24.9 Verifying Private and Public Keys Choose one of these values: `0` - do not check the certificate against a CRL. `1` - check the certificate against a CRL. This is the default.
revocation unknown	Determines the action to take when an ambiguous status is returned when checking a certificate against a CRL. In this case, it is not certain whether the certificate is valid or has a revoked status. See 24.9 Verifying Private and Public Keys for more information. Choose one of these values: `ok` - treat the certificate as valid. `revoked` - treat the certificate as revoked. This is the default.
sendencrypt cert	Controls whether the certificate of a public key that is used to encrypt an outgoing message is checked against a CRL before using it. See 24.9 Verifying Private and Public Keys Choose one of these values: `0` - do not check the certificate against a CRL. `1` - check the certificate against a CRL. This is the default.

sendencrypt certrevoked	Determines the action to take if the certificate of a public key that is used to encrypt an outgoing message is revoked. See 24.9 Verifying Private and Public Keys for more information. Choose one of these values: `allow` - use the public key. `disallow` - do not use the public key. This is the default.
sendsigncert	Controls whether a public key's certificate is checked against a CRL to determine if a private key can be used to create a digital signature for an outgoing message. (A private key is used for a digital signature but it cannot be checked against a CRL, so the certificate of the public key associated with the private key is checked against the CRL.) See 24.9 Verifying Private and Public Keys for more information. Choose one of these values: `0` - do not check the certificate against a CRL. `1` - check the certificate against a CRL. This is the default.
sendsigncert revoked	Determines the action to take when it is determined that a private key has a revoked status. (A private key is used to create a digital signature for a message but it cannot be checked against a CRL, so the certificate of the public key associated with the private key is checked against the CRL. If the public key certificate is revoked, then it's corresponding private key is also revoked.) See 24.9 Verifying Private and Public Keys for more information. Choose one of these values: `allow` - use the private key with a revoked status. `disallow` - do not use the private key with a revoked status. This is the default.
sslrootcacertsurl	Specifies the distinguished name and the LDAP directory information to locate the certificates of valid CAs which are used to verify the Messaging Server's SSL certificates. This is a required parameter when SSL is enabled in the Messaging Server. See 24.7 Securing Internet Links With SSL for more information. If you have SSL certificates for a proxy server that receives all requests from client application, the CA certificates for those SSL certificates must also be located in the LDAP directory pointed to by this parameter. You can also optionally add the DN and password that has access to the URL. Syntax: `crlmappingurl` *URL* [|*URLDN* | *URLpassword*] Example: `sslrootcacertsurl==ldap://mail.siroe.com:389/cn=SSL Root CA Certs,ou=people,o=siroe.com,o=isp? cacertificate;binary?base? (objectclass=certificationauthority)|cn=Directory Manager | pAsSwOrD`

timestampdelta	Specifies a time interval, in seconds, that is used to determine whether a message's sent time or received time is used when checking a public key's certificate against a CRL. The parameter's default value of zero directs Communications Express Mail to always use the received time. See 24.9.6 Determining Which Message Time to Use for more information. Example: `timestampdelta==360`
trustedurl	Specifies the distinguished name and LDAP directory information to locate the certificates of valid CAs. This is a required parameter. You can also optionally add the DN and password that has access to the URL. Syntax: `crlmappingurl` *URL* [|*URL DN* | *URL password*] Example: `trustedurl==ldap://mail.siroe.com:389/cn=Directory Manager, ou=people, o=siroe.com,o=ugroot?cacertificate?sub? (objectclass=certificationauthority)|cn=Directory Manager | pAsSwOrD`
usercertfilter	Specifies a filter definition for the primary, alternate, and equivalent email addresses of a Communications Express Mail user to ensure that all of a user's private-public key pairs are found when they are assigned to different mail addresses. This parameter is required and has no default values.

Top

Messaging Server Options

To set the three Messaging Server options that apply to S/MIME, do the following on the machine where the Messaging Server is installed.

To Set Messaging Server Options that Apply to S/MIME

1. Log in as root. Then enter:

   ```
   # cd msg-svr-base/sbin
   ```

   where *msg-svr-base* is the directory where Messaging Server is installed.

2. Set the Messaging Server options, described in the following table, as desired for your system. Use the `configutil` utility to set them. Unless stated otherwise, an option is not required to be set.

Parameter	Purpose
`local.webmail.cert.enable`	Controls whether the process that handles CRL checking should do CRL checking. `0` - the process does not check a certificate against a CRL. This is the default. `1` - the process checks a certificate against a CRL. When set to `1`, ensure that the `crlenable` parameter in the `smime.conf` file is set to `1`.
`local.webmail.cert.port`	Specifies a port number on the machine where the Messaging Server runs to use for CRL communication. This port is used locally for that machine only. The value must be greater than 1024. The default is `55443`. This is a required option if the default port number is already in use.
`local.webmail.smime.enable`	Controls whether the S/MIME features are available to Communications Express Mail users. Choose one of these values: `0` - the S/MIME features are unavailable for Communications Express Mail users even though the system is configured with the correct software and hardware components. This is the default. `1` - the S/MIME features are available to Communications Express Mail users who have permission to use them. Example: `configutil -o local.webmail.smime.enable -v 1`

Top

Securing Internet Links With SSL

The Messaging Server supports the use of the Secure Socket Layer (SSL) for Internet links affecting Communications Express Mail, as summarized in the following table.

Link Between:	Description
Messaging Server and Communications Express Mail	Securing this link with SSL requires administrative work for the Messaging Server. The Communications Express Mail user must use the HTTPS protocol, rather than HTTP, when entering the URL information for the Messaging Server in their browser. See 24.7.1 Securing the Link Between Messaging Server and Communications Express Mail
Messaging Server and S/MIME applet	When checking public keys certificates against a CRL, the S/MIME applet must communicate directly with the Messaging Server. Securing this link with SSL requires administrative work for the Messaging Server in addition to setting `sslrootcacertsurl` and `checkoverssl` in the `smime.conf` file. See 24.7.2 Securing the Link Between the Messaging Server and S/MIME Applet

Top

Securing the Link Between Messaging Server and Communications Express Mail

The Messaging Server supports the use of Secure Socket Layer (SSL) for the Internet link between it and Communications Express Mail. Once you have set up Messaging Server for SSL, configure Communications Express for SSL See *Sun Java System Communications Express 6.3 Administration Guide*. A Communications Express Mail user specifies the Communications Express URL in their browser with the HTTPS protocol:

```
HTTPS://hostname.domain:
secured_port
```

instead of the HTTP protocol (`HTTP://`*hostname*`.`*domain*`:`*unsecure_port*). When the Communications Express login window displays, the user sees a lock icon in a locked position at the bottom of their window to indicate they have a secure link.

See Configuring Encryption and Certificate-Based Authentication for SSL configuration information for Messaging Server.

Top

Securing the Link Between the Messaging Server and S/MIME Applet

When checking the certificate of a public key against a CRL, the S/MIME applet must communicate directly with the Messaging Server.

To Secure the Communications Link with SSL

1. Do the administrative tasks to configure the Messaging Server for SSL. See Configuring Encryption and Certificate-Based Authentication.
2. Set the `sslrootcacertsurl` parameter in the `smime.conf` file to specify the information to locate the root SSL CA certificates. These CA certificates are used to verify the Messaging Server's SSL certificates when the SSL link is established between the Messaging Server and the S/MIME applet.
3. Set the `checkoverssl` parameter in the `smime.conf` file to `1`. This Messaging Server option determines whether SSL is used for the link between the Messaging Server and the S/MIME applet. Regardless of how a Communications Express Mail user specifies the URL for the

Messenger Server (HTTP or HTTPS), the link between the Messaging Server and the S/MIME applet is secured with SSL when `checkoverssl` is set to `1`.

> **Note**
> A proxy server can be used between the Messaging Server and client applications such as Communications Express Mail. See 24.9.4 Proxy Server and CRL Checking using a proxy server with and without a secured communications link.

Top

Key Access Libraries for the Client Machines

Whether your mail users keep their private-public key pairs and certificates on a smart card or in a local key store of their browsers, key access libraries must be present on the client machines to support the storage methods.

The libraries are supplied by vendors of the smart cards and browsers. You must ensure that the correct libraries are on the client machines and specify the library name or names with the appropriate platform parameter in the `smime.conf` file. The parameters choices are:

- `platformwin` for Microsoft Windows running on a PC.

You can specify only the libraries you know are installed on the client machines or you can specify all the library names for a given platform and vendor if you are not sure what is installed. If the S/MIME applet does not find the library it needs among the names you specify, the S/MIME features do not work.

The syntax to specify one or more library file names is:

```
platform_parameter==vendor:library=
library_name;...
```

where:

platform_parameter is the parameter name for the platform of the client machine where Communications Express Mail is accessed. Choose one of these names: `platformwin`

vendor specifies the vendor of the smart card or browser. Choose one of these literals:

`cac` (for an ActivCard or NetSign smart card)

`capi` (for Internet Explorer with CAPI)

`mozilla` (for Mozilla with Network Security Services)

library_name specifies the library filename. See Table 24-4 for the library name for your vendor and operating system.

Special Libraries for the Client Machines

Smart Card or Browser Vendor	Operating System	Library Filename
	Windows	`acpkcs211.dll`
Internet Explorer with Cryptographic Application Programming Interface (CAPI)	Windows	`capibridge.dll`
	Windows	`softokn3.dll`
	Windows	core32.dll

Top

Example

The following example specifies one smart card library and one Internet Explorer library, and one Mozilla library for a Windows platform:

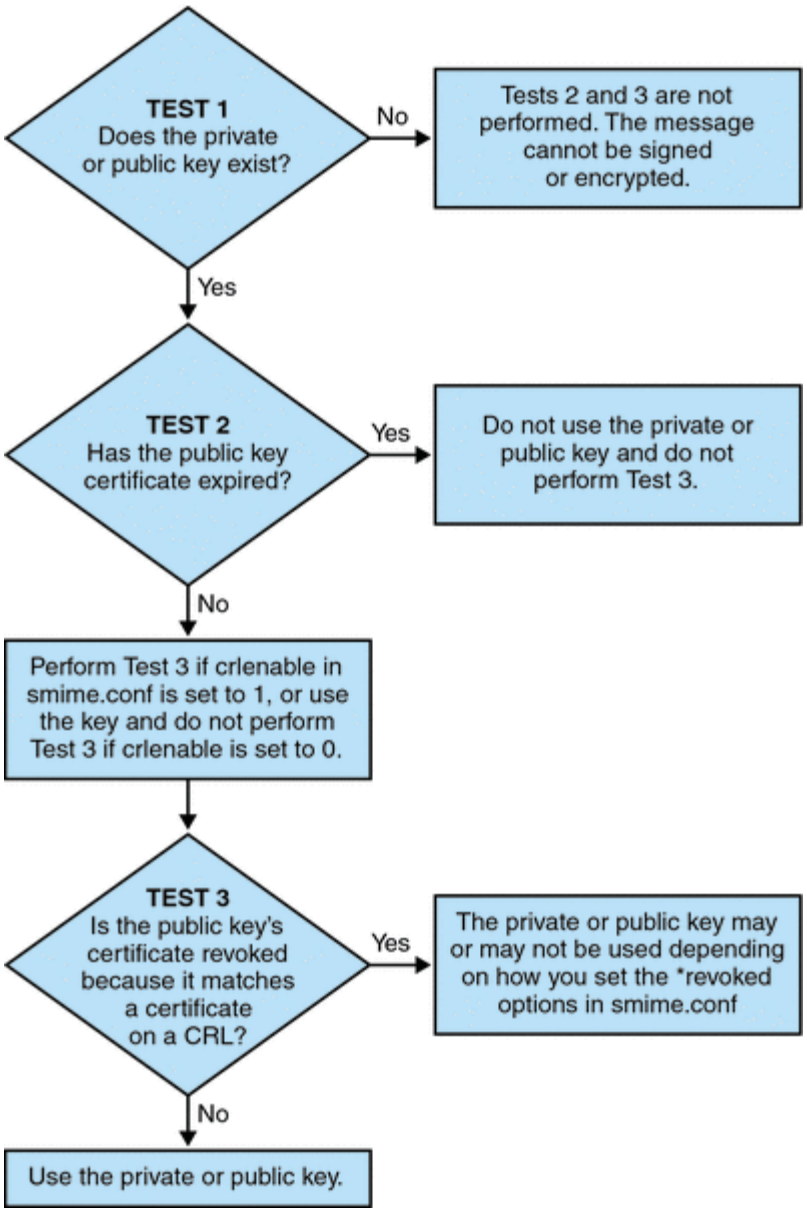
```
platformwin==CAC:library=acpkcs211.dll;CAPI:library=capibridge.dll;
MOZILLA:library=softokn3.dll;
```

Top

Verifying Private and Public Keys

Before Communications Express Mail uses a private or public key, it must pass the verification tests shown in Figure 24-2. The remainder of this section describes the details of checking a public key's certificate against a CRL.

Verifying Private and Public Keys.



Top

Finding a User's Private or Public Key

When a Communications Express Mail user has multiple private-public key pairs and multiple email addresses (primary, alternate, or alias addresses), it is possible that their keys are associated among their addresses. In this case, it is important that the S/MIME applet finds all the keys for verification purposes. Use the `usercertfilter` parameter in the `smime.conf` file to define a filter that creates a list of mail addresses for a key's owner at the time the public key's certificate is checked against a CRL. See `usercerfilter` in 24.5 Parameters of the smime.conf File for more information.

Top

When is a Certificate Checked Against a CRL?

A certificate revocation list, or CRL, is a list of revoked certificates maintained by the CA who issues the key pairs and certificates. When CRL checking is enabled, it causes the system to check the CRL whenever a certificate request has been made to see whether or not that certificate has been revoked.

When `crlenable` is set to `1` in the `smime.conf` file, a CRL test is performed after an unexpired key is found. The public key's certificate is checked against a CRL. There can only be one CRL for each CA, however the same CRL can be located in different places.

Checking a certificate against a CRL is done by the Messaging Server after the S/MIME applet sends it a request to do so. A public key certificate is used to validate a public key. Because a private key is kept secret, only used by the person who owns it, a private key cannot be checked directly against a CRL. To determine if a private key is good, the public key certificate of the key pair is used. When the public key's certificate passes the CRL test, the associated private key passes the test too.

Revocation of a certificate can happen for a variety of reasons, such as its owner has left your organization or lost the smart card.

There are three situations for checking a certificate against a CRL:

- When an outgoing message is signed

  The S/MIME applet always does this check unless you set `sendsigncert` to `0` or `crlenable` to `0`.
- When an incoming signed message is read

  The S/MIME applet always does this check unless you set `readsigncert` to `0` or `crlenable` to `0`.
- When an outgoing message is encrypted

  The S/MIME applet always does this check unless you set `sendencryptcert` to `0` or `crlenable` to `0.`

Top

Accessing a CRL

A certificate contains zero or more URLs, known as distribution points, that are used by Messaging Server to locate a CRL. If the certificate does not have a CRL URL, it cannot be checked against a CRL and the private or public key is used to sign or encrypt a message without knowing its true status.

If Messaging Server fails to locate or gain access to a CRL after trying all the URLs available to it, the status of the certificate is treated as unknown. Whether a private or public key with an unknown status is used is determined by the setting of `revocationunknown`.

While only one CRL for each CA is supported, there can be multiple copies of the same CRL in different locations, reflected in different URLs among a user's public key certificates. Messaging Server tries all the URL locations for a certificate until it gains access to the CRL.

You can manage multiple copies of a CRL for optimum access by periodically downloading the current CRL from the CA to a place where you want it. While you cannot change the URLs embedded in the certificates, you can redirect Messaging Server to use new CRL locations by mapping the URLs in a certificate to a new URL containing the CRL information. Create a list of one or more mapping definitions in the LDAP directory (see crlmappingurl in Table 24-3) with this syntax:

```
msgCRLMappingRecord=url_in_certificate==
new_url[|url_login_DN|url_login_password]
```

url_in_certificate is the URL in the certificate containing the old information to locate the CRL. new_url is the new URL containing the new CRL information. *url_login_DN* and *url_login_password* are the DN and

password of the entry allowed access to new_url. Both are optional, and if specified, will be used for the new URL access only.

If the DN and password fails, LDAP access is denied and no retry with other credentials is attempted. These login credentials are only valid for LDAP URLs. If you use `crlurllogindn` and `crlurlloginpw` in `smmime.conf`, then you don't need to specify the login DN and password in the mapping record. See 24.4.3 Accessing LDAP for Public Keys, CA certificates and CRLs Using Credentials

Only one layer of mapping is allowed. Different URLs in the certificates can be mapped to the same new URL, but you cannot assign a certificate URL to multiple new URLs. For example, the following mapping list is not valid:

```
msgCRLMappingRecord=URL12==URL45
msgCRLMappingRecord=URL12==URL66
msgCRLMappingRecord=URL12==URL88
msgCRLMappingRecord=URL20==URL90
msgCRLMappingRecord=URL20==URL93
```

The next example is a correct mapping list:

```
msgCRLMappingRecord=URL12==URL45
msgCRLMappingRecord=URL14==URL66
msgCRLMappingRecord=URL88==URL66
msgCRLMappingRecord=URL201==URL90
msgCRLMappingRecord=URL202==URL93
```

Once you have created the mapping definitions in your LDAP directory, use `crlmappingurl` in the `smime.conf` file to specify the directory information to locate them. See 24.5 Parameters of the smime.conf File.

Top

Proxy Server and CRL Checking

If your system uses a proxy server between client applications and the Messaging Server, CRL checking can be blocked despite the fact that you correctly configured the S/MIME applet to perform CRL checking. When this problem occurs, users of Communications Express Mail receive error messages alerting them to revoked or unknown status for valid key certificates.

The following conditions cause the problem:

- CRL checking is requested with these configuration values:
  - `crlenable` parameter in the `smime.conf` file is set to `1`
  - `local.webmail.cert.enable` option of Messaging Server is set to `1`
- The communications link between the S/MIME applet and the proxy server is not secured with SSL, but the S/MIME applet is expecting a secured link because the `checkoverssl` parameter in the `smime.conf` file is set to `1`

To solve this problem, you can:

1. Set up the communications link between the client machines and proxy server as a secured link with SSL and leave all the configuration values as they are. Or,
2. Leave the communications link unsecured and set `checkoverssl` to `0`.

For more information see 24.7 Securing Internet Links With SSL.

Top

Using a Stale CRL

Checking a certificate against a CRL is done by the Messaging Server after the S/MIME applet sends it a request to do so. Rather than download a CRL to memory each time a certificate is checked, Messaging Server downloads a copy of the CRL to disk and uses that copy for certificate checking. Every CRL has a next-update field which specifies the date after which a newer CRL version should be used. The next-update date can be viewed as an expiration date or time limit for using the CRL. A CRL that is past it's next-update date is considered old or stale and triggers Messaging Server to download the latest version of the CRL the next time a certificate is checked.

Every time the S/MIME applet requests that a certificate be checked against a CRL, the Messaging Server does the following:

1. Compares the current date to the next-update date of the CRL.
2. If the CRL is stale, the Messaging Server downloads the latest version of the CRL to replace the stale CRL on disk and checking proceeds. However, if a newer CRL cannot be found or cannot be downloaded, the value of `crlusepastnextupdate` in the `smime.conf` file is used to determine what to do.
3. If `crlusepastnextupdate` is set to `0`, the stale CRL is not used and the certificate in question has an ambiguous status. The S/MIME applet uses the value of `revocationunknown` in `smime.conf` to determine what to do next:
    a. If `revocationunknown` is set to `ok`, the certificate is treated as valid and the private or public key is used to sign or encrypt a message.
    b. If `revocationunknown` is set to `revoked`, the certificate is treated as invalid, the private or public key is not used to sign or encrypt a message, and a pop-up error message alerts the mail user that the key cannot be used.

    If `crlusepastnextupdate` is set to `1`, the S/MIME applet continues to use the stale CRL which causes no interruption of processing within Communications Express Mail, however a message is written to the Messaging Server log file to alert you to the situation.

This sequence of events continues to occur as certificates are checked against the CRL. As long as the Messaging Server can download a newer version of the CRL in a timely manner, and depending on the settings in the `smime.conf` file, mail processing proceeds without interruption. Check the Messaging Server log periodically for repeated messages that indicate a stale CRL is in use. If a newer CRL cannot be downloaded, you need to investigate why it is inaccessible.

Top

Determining Which Message Time to Use

The `timestampdelta` parameter is used primarily for these purposes:

1. To handle the situation of a message that takes a long time to arrive at its destination. For this case, the sender's key might be treated as an invalid key despite the fact that the key was valid when the message was sent.
2. To limit the trust in a message's sent time because sent times can be faked.

There are two times associated with every message:

- The time when the message was sent, as found in the Date line of the message header detail
- The time when the message arrives at its destination, as found in the last Received line of the message header detail

> **Note**
> View the message header detail by clicking the triangle icon at the right hand side of a message's From field.

A certificate that was valid when a message was sent can be revoked or expired by the time the message reaches its destination. When this happens, which time should be used when checking the validity of the certificate, the sent time or the received time? Using the sent time would verify that the certificate was valid when the message was sent. But always using the sent time does not take into account the fact that it might take a long time for a message to arrive at its destination, in which case it would be better to use the received time.

You can influence which time to use for CRL checking by using the `timestampdelta` parameter in the `smime.conf` file. Set this parameter to a positive integer, representing seconds. If the received time minus the value of `timestampdelta` is a time before the sent time, the sent time is used. Otherwise, the received time is used. The smaller the value of `timestampdelta`, the more often the received time is used. When `timestampdelta` is not set, the received time is always used. See `timestampdelta` in Table 24-3.

Top

Trouble Accessing a CRL

For a variety of reasons, such as network or server problems, a CRL might be unavailable when Messaging Server attempts to check a certificate against it. Rather than let the Messaging Server spend its time constantly trying to gain access to the CRL, you can use the `crlaccessfail` parameter in the `smime.conf` file to manage how often it attempts to access the CRL, freeing up the Messaging Server for other tasks.

Define the following with `crlaccessfail`:

- How many failed attempts are counted (an error message is written to the Messaging Server log after each failed attempt)
- Over what period of time the failed attempts are counted
- How long to wait before attempting a new cycle of accessing the CRL

See `crlaccessfail` in Table 24-3 for the parameter's syntax and an example.

Top

When a Certificate is Revoked

When a public key's certificate does not match any entry on the CRL, the private or public key is used to sign or encrypt an outgoing message. When a certificate matches an entry on the CRL or the certificate's status is unknown, a private or public key is considered revoked. By default Communications Express Mail does not use a key with a revoked certificate to sign or encrypt an outgoing message. If the private key of a signed message is revoked by the time the recipient reads the message, the recipient receives a warning message indicating that the signature should not be trusted.

If desired, you can change the various default policies for all revoked certificates with the following parameters in the `smime.conf` file:

- Set `sendsigncertrevoked` to `allow` to sign an outgoing message with a private key that is considered revoked because its public key's certificate is revoked
- Set `sendencryptcertrevoked` to `allow` to encrypt an outgoing message with a public key that has a revoked certificate

- Set `revocationunknown` to `ok` to treat a certificate as valid whose status is unknown; the private or public key is used to sign or encrypt an outgoing message

Top

Granting Permission to Use S/MIME Features

Permission to use the various mail services available through Communications Express Mail can be given or denied with LDAP filters. A filter is defined with the `mailAllowedServiceAccess` or `mailDomainAllowedServiceAccess` LDAP attributes. Generally speaking, a filter works in one of three ways:

- Permission to given to all users for all services when no filter is used
- Permission is explicitly given to a list of users for specified service names (a plus sign (+) precedes the service name list)
- Permission is explicitly denied to a list of users for specified service names (a minus sign precedes the service name list)

The required mail service names for S/MIME are `http`, `smime`, and `smtp`. If you need to restrict the use of S/MIME among Communications Express Mail users, use the appropriate LDAP attribute syntax and service names to create a filter. The attributes are created or modified with LDAP commands.

Top

S/MIME Permission Examples

1. The following examples block access to the S/MIME features for one Communications Express Mail user:

```
mailAllowedServiceAccess
mailAllowedServiceAccess: -smime:*$+imap,pop,http,smtp:*
```

or

```
mailAllowedServiceAccess: +imap,pop,http,smtp:*
```

2. The following examples block access to the S/MIME features for all Communications Express Mail users in a domain:

```
mailDomainAllowedServiceAccess: -smime:*$+imap:*$+pop:*$+smtp:*$+http:*
```

or

```
mailDomainAllowedServiceAccess: +imap:*$+pop:*$+smtp:*$+http:*
```

See Filter Syntax for more information.

Top

Managing Certificates

Most of the following examples use the `ldapsearch` and `ldapmodify` commands to search an LDAP directory for user keys and certificates. These commands are provided with Directory Server. See the *Sun ONE Directory Server Resource Kit Tools Reference*, Release 5.2, for more information about the commands.

Top

CA Certificates in an LDAP Directory

This example adds a certificate for a certificate authority to an LDAP directory. The directory structure for these certificates already exists. The certificate and the LDAP entries where it belongs are entered into an `.ldif` file named `add-root-CA-cert.ldif`. All text is entered into the file in ASCII text except for the certificate information, which must be entered as Base64 encoded text:

```
dn: cn=SMIME Admin,ou=people,o=demo.siroe.com,o=demo
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: certificationAuthority
cn: RootCACerts
sn: CA
authorityRevocationList: novalue
certificateRevocationList: novalue
cacertificate;binary:: MFU01JTUUEjAQBgNVBAsTCU1zZ1NlcnZlcjcMBoGA1UEAxMTydG
QGEwJVUzEOMAwGA1UEMFUJTUUxEjAQBgNVBAsTCU1zZ1NlcnZlcjEMBoGA1UEAxMTQ2VydG
aFw0wNjAxMwODAwMDBaM267hgbX9FExCzAJByrjgNVBAk9STklBMQwCgYDVQQVHR8EgaQwg
YTAlVMRMQYDVQQIEwpDQUxJRk9STklBMQwwCgYDVQQKEwww3ltgYz11lzAdBgNVBpYSE9Vc
5yZWQaddWlm899XBsYW5ldC5jb20wgZ8wDQYJoGBAK1mUTy8vvnOFg4mlHjkghytQUR1k8l
5mvWRf77ntm5mGXRD3XMU4OciUq6zUfIg3ngvxlLyERTIqjUS8HQU4R5pvj+rrVgsAGjggE
+FNAJmtOV2A3wMyghqkVPNDP3Aqq2fkcn4va3C5nRNAYxNNVE84JJ0H3jyPDXhMBlQU6vQn
weMBAAjggEXMIIBEzARBglghkgBhCAQEEBApqlSai4mfuvjh02SQkoPMNDAgTwMB8GA1UdI
QYMBaAEd38IK05AHreiU9OYc6vNMOwZMIGsBgNVHR8EgaQwgaEwb6BtoGuGaWxkYXA6Lyht
bmcucmVkLmlbGFuZXQuY29tL1VJD1DXJ0aWZpY2F0ZSBNYW5hZ2VyLE9VPVBlb3BsZSxPPW
aWxxYT9jZXJ0aZpY2jdu2medXRllkghytQURYFNrkuoCygKoYoaHR0cDovL3Bla2kghytQU
Zy5yZWQuaXBsYW5lC5jb20vcGVranLmNybDAeBgNVHREEFzAVgRNwb3J0aWEuc2hhb0BzdW
4uY29tMA0GCxLm78freCxS3Pp078jyTaDci1AudBL8+RrRUQvxsMJfZeFED+Uuf10Ilt6kw
Tc6W5UekbirfEZGAVQIzlt6DQJfgpifGLvtQ60Kw==
```

The CA's certificate is added to the LDAP directory with an `ldapmodify` command:

```
# ldapmodify -a -h demo.siroe.com -D "cn=Directory Manager" -w mypasswd -v
 -f add-root-CA-cert.ldif
```

The value of the `trustedurl` parameter in `smime.conf` specifies the location of the CA certificates in the LDAP directory. For Example 1, `trustedurl` is set to:

```
trustedurl==ldap://demo.siroe.com:389/cn=SMIME Admin, ou=people,
o=demo.siroe.com,o=demo?cacertificate;binary?sub?
(objectclass=certificationAuthority)
```

Top

Public Keys and Certificates in an LDAP Directory

This example demonstrates adding a mail user's public key and certificate to the LDAP directory. It assumes the mail user already exists in the LDAP directory. The key and certificate, and the LDAP entries where it belongs, are entered into an `.ldif` file named `add-public-cert.ldif`. All text is entered into the file as ASCII text except for the key and certificate information, which must be entered as Base64 encoded text.

```
dn: uid=JohnDoe,ou=People, o=demo.siroe.com,o=demo
changetype: modify
replace: usercertificate
usercertificate;binary:: MFU01JTUUxEjAQBgNVBAsT1zZ1NlcnZlcjMBoGA1UEAxMTydG
QGEwJVUzEAwGA1hMFU01JTUUxEjAQBgNVBAsTCU1zZ1NlcnZlcjEcMBoGA1UEAxMTQ2VydG
aFw0wNjAxMTODAwaM267hgbX9FExCzAJBgwyrjgNVBAk9STklBMQwwCgYDVQQVHR8EgaQwg
AlVzMRMwEQYDVQQIDQUxJRk9STklBMQwwCgYDVQQKEwww3ltgoOYz11lzAdBgNVBpYSE9Vc
5yZWaddiiWlm899XBsYW5ldb20wgZ8wDQYJoGBAK1mUTy8vvO2nOFg4mlHjkghytQUR1k8l
5mvgcWL77ntm5mGXRD3XMU4OcizUfIg3ngvxlLKLyERTIqjUS8HQU4R5pvj+rrVgsAGjggE
+FG9NAqtOV2A3wMyghqkVPNDP3Aqq2BYfkcn4va3RNAYxNNVE84JJ0H3jyPDXhMBlQU6vQn
1NAgMBGjggEXMIIBEzARBglghkgBhvhCAQEEBApqlSai4mfuvjh02SQMNDAgTwMB8GA1UdI
QYMBaEd38IK05AHreiU9OYc6v+ENMOwZMIGsBgNVHR8EgaQwgaEwb6BuGaWxkYXA6Lyht74
tpbmcmVkLmlwbGFuZXQuY29tL1VJRD1DZXJ0aWZpY2F0ZSBNYW5hZ2V9VPVBlb3BsZSxPPW
1haWxT9jZXJ0aWZpY2jdu2medXRllHjkghytQURYFNrkuoCygKoYoaHDovL3Bla2kghytQU
luZy5WQuaXBsYW5ldC5jb20vcGVraW5nLmNybDAeBgNVHREEFzAVgRNw0aWEuc2hhb0BzdW
4uY29A0GCxLm78UfreCxS3Pp078jyTaDv2ci1AudBL8+RrRUQvxsMJfZD+Uuf10Ilt6kwhm
Tc6W5UekbirfEZGAVQIzlt6DQJfgpifGLvtQ60Kw==
```

The `ldapmodify` command is used to add the public key and certificate to the LDAP directory:

```
# ldapmodify -a -h demo.siroe.com -D "cn=Directory Manager" -w mypasswd -v
 -f add-public-cert.ldif
```

The value of the `certurl` parameter in `smime.conf` specifies the location of the public keys and their certificates in the LDAP directory. For Example 2, `certurl` is set to:

```
certurl==ldap://demo.siroe.com:389/ou=people, o=demo.siroe.com,
o=demo?userCertificate;binary?sub?
```

Top

Verifying That Keys and Certificates Exist in the LDAP Directory

The following examples demonstrate searching an LDAP directory for CA certificates and public keys

and their certificates.

Top

Searching for One CA Certificate

In the following example, the base DN defined by the `-b` option, `cn=SMIME admin, ou=people,o=demo.siroe.com,o=demo objectclass=*`, describes one CA certificate in the LDAP directory. If found in the directory, `ldapsearch` returns information about the certificate to the `ca-cert.lidf` file.

```
# ldapsearch -L -h demo.siroe.com -D "cn=Directory Manager" -w mypasswd -b
"cn=SMIME admin, ou=people,o=demo.siroe.com,o=demo" "objectclass=*"
> ca-cert.ldif
```

The example below shows the search results in the `ca-cert.ldif` file. The format of the file's contents is a result of using the `-L` option of `ldapsearch`.

```
# more ca-cert.ldif
dn: cn=SMIME admin,ou=people,o=demo.siroe.com,o=demo
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: certificationAuthority
cn: RootCACerts
cn: SMIME admin
sn: CA
authorityRevocationList: novalue
certificateRevocationList: novalue
cacertificate;binary:: MFU01JTUUxEjAQBgNVBAsTCU1zZNlcnZlcjcMBoGA1UEAxMTydG
QGEwJVEOMAwGA1UEChMFU0UUxEjAQBgNVBAsTCU1zZ1NlcnZlcjEcMBoGA1UEAxMTQ2VydG
aFw0jAxMTIwODAwMDBaM267X9FExCzAJBgwyrjgNVBAk9STklBMQwwCgYDVQQVHR8EgaQwg
YlVzMRMwEQYDVQQIEwpDQUx9STklBMQwwCgYDVQQKEwww3ltgoOYz11lzAdBgNVBpYSE9Vc
5yQuaddiiWlm899XBsYW5ljb20wgZ8wDQYJoGBAK1mUTy8vvO2nOFg4mlHjkghytQUR1k8l
5mcWRfL77ntm5mGXRD3XMciUq6zUfIg3ngvxlLKLyERTIqjUS8HQU4R5pvj+rrVgsAGjggE
+FNAJmqtOV2A3wMyghqkDP3Aqq2BYfkcn4va3C5nRNAYxNNVE84JJ0H3jyPDXhMBlQU6vQn
1NABAAGjggEXMIIBEzglghkgBhvhCAQEEBApqlSai4mfuvjh02SQkoPMNDAgTwMB8GA1UdI
QYMAFEd38IK05AHreOYc6v+ENMOwZMIGsBgNVHR8EgaQwgaEwb6BtoGuGaWxkYXA6Lyht74
tpbucmVkLmlwbGFuZY29tL1VJRD1DZXJ0aWZpY2F0ZSBNYW5hZ2VyLE9VPVBlb3BsZSxPPW
1haWYT9jZXJ0aWZpdu2medXRllHjkghytQURYFNrkuoCygKoYoaHR0cDovL3Bla2kghytQU
luZyZWQuaXBsYW5ldb20vcGVraW5nLmNybDAeBgNVHREEFzAVgRNwb3J0aWEuc2hhb0BzdW
4uYtMA0GCxLm78Ufre3Pp078jyTaDv2ci1AudBL8+RrRUQvxsMJfZeFED+Uuf10Ilt6kwhm
Tc6W5UekbirfEZGAVQIzlt6DQJfgpifGLvtQ60Kw==
```

Top

Searching for a Several Public Keys

In the following example, the base DN defined by the `-b` option, `o=demo.siroe.com,o=demo objectclass=*`, is such that all public keys and certificates found at and below the base DN in the LDAP directory are returned to the file `usergroup.ldif`:

```
# ldapsearch -L -h demo.siroe.com -D "cn=Directory Manager" -w mypasswd
-b "o=demo.siroe.com,o=demo" "objectclass=*" > usergroup.ldif
```

Top

Searching for One Public Key

In the following example, the base DN defined by the `-b` option, `uid=JohnDoe, ou=people,o=demo.siroe.com,o=demo objectclass=*`, describes one public key and its certificate in the LDAP directory:

```
# ldapsearch -L -h demo.siroe.com -D "cn=Directory Manager" -w mypasswd -b
"uid=JohnDoe, ou=people,o=demo.siroe.com,o=demo" "objectclass=*" >
public-key.ldif
```

The example below shows the search results in the `public-key.ldif` file. The format of the file's contents is the result of using the `-L` option of `ldapsearch`.

```
# more public-key.ldif
dn: uid=sdemo1, ou=people, o=demo.siroe.com, o=demo
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: siroe-am-managed-person
objectClass: inetOrgPerson
objectClass: inetUser
objectClass: ipUser
objectClass: userPresenceProfile
objectClass: inetMailUser
objectClass: inetLocalMailRecipient
objectClass: icsCalendarUser
objectClass: sunUCPreferences
mail: JohnDoe@demo.siroe.com
mailHost: demo.siroe.com
.
.
uid: JohnDoe
.
.
mailUserStatus: active
inetUserStatus: active
.
.
usercertificate;binary:: MFU01JTUUxEjAQBgNBAsTCU1zZ1NlcnZjcMBoGA1UEAxMTydG
QGEwJEOwGA1UEChMFU01JTUUxEjAQBgNVBAsTCU1zZ1NlcnZlcjEcMBoGA1UEAxMTQ2VydG
aFw0MTIwODAwMDBaM267hgbX9FExCzAJBgwyrjgNVBAk9STklBMQwwCgYDVQQVHR8EgaQwg
YTAlVEQYDVQQIEwpDQUxJRk9STklBMQwwCgYDVQQKEwww3ltgoOYz11lzAdBgNVBpYSE9Vc
5yZWQdWlm899XBsYW5ldC5jb20wgZ8wDQYJoGBAK1mUTy8vvO2nOFg4mlHjkghytQUR1k8l
5mvgc7ntm5mGXRD3XMU4OciUq6zUfIg3ngvxlLKLyERTIqjUS8HQU4R5pvj+rrVgsAGjggE
+FG9NmV2A3wMyghqkVPNDP3Aqq2BYfkcn4va3C5nRNAYxNNVE84JJ0H3jyPDXhMBlQU6vQn
1NAgMAgEXMIIBEzARBglghkgBhvhCAQEEBApqlSai4mfuvjh02SQkoPMNDAgTwMB8GA1UdI
QYMBaEdK05AHreiU9OYc6v+ENMOwZMIGsBgNVHR8EgaQwgaEwb6BtoGuGaWxkYXA6Lyht74
tpbucmVkwbGFuZXQuY29tL1VJRD1DZXJ0aWZpY2F0ZSBNYW5hZ2VyLE9VPVBlb3BsZSxPPW
1haxYT9jZaWZpY2jdu2medXRllHjkghytQURYFNrkuoCygKoYoaHR0cDovL3Bla2kghytQU
luZyZWQuaYW5ldC5jb20vcGVraW5nLmNybDAeBgNVHREEFzAVgRNwb3J0aWEuc2hhb0BzdW
4u9tMA0GC78UfreCxS3Pp078jyTaDv2ci1AudBL8+RrRUQvxsMJfZeFED+Uuf10Ilt6kwhm
Tc6W5UekbirfEZGAVQIzlt6DQJfgpifGLvtQ60Kw==
.
.
```

Top

Network Security Services Certificates

Various certificates used for Network Security Services (NSS) are stored in their own database, which is not an LDAP database. Two utilities, `certutil` and `crlutil`, are provided with Messaging Server to store the certificates and associated CRLs in the database. You can also use these utilities to search the database.

See the Sun Java System Directory Server Administration Guide/819-0992.pdf) for more information about `certutil`. Use the help text that comes with `crlutil` for more information about that utility (view the online help of both utilities by executing them without arguments).

Top

Communications Express S/MIME End User Information

This section consists of information intended for the end user. It contains the following subsections:

Top

Logging In for the First Time

When mail users log in to Communications Express Mail for the first time, they encounter special prompts relating to the S/MIME applet.

Top

Prompts for Windows

When logging in to Communications Express Mail for the first time on Windows 98, 2000 or XP, the following prompts display:

1. If the Java 2 Runtime Environment (JRE) is not installed on your computer (client machine), you receive a prompt looking something like this:

   ```
   Do you want to install and run "Java Plug-in 1.4.2_03 signed on 11/20/03
   and distributed by Sun Microsystems, Inc."?Publisher authenticity
   verified by: VeriSign Class 3 Code Signing 2001 CA
   ```

   Click Yes and follow the subsequent prompts to install JRE.

   > **Note**
   > If you desire English language support and also want to read incoming S/MIME messages that contain non-Latin characters, such as Chinese, the `charsets.jar` file must be in the `/lib` directory on your computer.
   >
   > To ensure that the `charsets.jar` file is installed in the `/lib` directory, use the custom installation to install the English version of JRE. During the installation process, select the "Support for Additional Languages" option.
   >
   > See 24.3.6 Multi-language Support for more information.

   Click Finish at the last installation prompt. Restart your computer and log in to Communications Express Mail again.

2. A prompt asking you:

   ```
   Do you want to trust the signed applet distributed by "Sun Microsystems,
   Inc."?Publisher authenticity verified by: Thawte Consulting cc
   ```

   Click one of the following responses:
   - Yes, to accept the S/MIME applet for this Communications Express Mail session. The prompt displays each time you log in.
   - No, to reject the S/MIME applet. You cannot use the S/MIME features.

- Always, to accept the S/MIME applet for this and all subsequent Communications Express Mail sessions. You will not see the prompt again.

3. A prompt asking you:

   ```
   Do you want to trust the signed applet distributed by "sun microsystems,
   inc."?Publisher authenticity verified by: VeriSign, Inc.
   ```

   Click one of the following responses:
   - Yes, to accept the S/MIME applet for this Communications Express Mail session. The prompt displays each time you log in.
   - No, to reject the S/MIME applet. You cannot use the S/MIME features.
   - Always, to accept the S/MIME applet for this and all subsequent Communications Express Mail sessions. You will not see the prompt again.

Top

Signature and Encryption Settings

There are initial signature and encryption settings that you can set to control whether all users' outgoing messages are:

- automatically signed, or
- automatically encrypted, or
- automatically signed and encrypted

The initial settings also control whether the signature and encryption checkboxes located at the bottom of a Communications Express Mail window and in the Options - Settings window are displayed as checked (feature turned on) or unchecked (feature turned off). Use the `alwaysencrypt` and `alwayssign` parameters in the `smime.conf` file to specify the initial settings.

Let your mail users know that they can change the initial settings for their mail messages. After they log in to Communications Express Mail, a user can temporarily override a setting for one message, or for all their messages on an on-going basis.

Table 24-5 summarizes the use of the checkboxes.

Signature and Encryption Checkboxes of Communications Express Mail

Text for Checkbox	Location	What Communications Express Mail User Does
Sign Message	At the bottom of the Communications Express Mail window used for composing, forwarding, or replying to a message.	• Check the box to sign the current message. • Uncheck the box not to sign the current message.
Encrypt Message	At the bottom of the Communications Express Mail window used for composing, forwarding, or replying to a message.	• Check the box to encrypt the current message. • Uncheck the box not to encrypt the current message.
Sign all outgoing Messages	In the Communications Express Mail Options - Settings window, under the Secure Messaging option.	• Check the box to sign all your messages automatically. • Uncheck the box not to sign all your messages automatically. Note: You can override the setting of "Sign all outgoing messages" on a message-by-message basis with the "Sign Message" checkbox.
Encrypt all outgoing Messages	In the Communications Express Mail Options - Settings window, under the Secure Messaging option.	• Check the box to encrypt all your messages automatically • Uncheck the box not to encrypt all your messages automatically. Note: You can override the setting of "Encrypt all outgoing messages" on a message-by-message basis with the "Encrypt Message" checkbox.

Top

Enabling the Java Console

A variety of operating messages can be written to the Java Console by the S/MIME applet as a Communications Express Mail user processes signed and encrypted messages. The Java Console messages can be helpful when troubleshooting a problem reported by a mail user. However, operating messages are only generated when the Java Console is enabled for the user by adding a `nswmExtendedUserPrefs` attribute to the `inetMailUser` object class of their LDAP entry. For example:

```
nswmExtendedUserPrefs: mesmimedebug=on
```

Do not enable the Java Console for all mail users all the time because this significantly decreases the performance of Communications Express Mail.

Chapter 29. Managing Logging

Managing Logging

This information provides overview information on the logging facilities for the Messaging Server MTA, the Message Store, and services. This information also provides procedures for how to manage these logging facilities.

Topics:

Overview of Logging

This section contains the following subsections:

What Is Logging and How Do You Use it?

Logging is the means by which a system provides you with time-stamped and labeled information about the system's services. Logging provides both a current snapshot of the system as well as a historical view.

By understanding and using Messaging Server log files, you can:

- Gather message statistics, such as message size, rate of message delivery, and how many messages are passing through the MTA
- Perform trend determination
- Correlate capacity planning
- Troubleshoot problems

For example, if your site needs to add more disk storage due to an increase in the number of users, you can use the Messaging Server log files to see what percentage your system demand has increased by and plan for the amount of new disk storage you need.

You can also use Messaging Server logs to understand what your messaging pattern looks like across one day. Understanding when your daily peak loads occur helps you conduct capacity planning.

Logging is also helpful for troubleshooting user problems. For example, if a user isn't receiving expected mail messages, you can use the Messaging Server logging facilities to trace the user's mail messages. In so doing, you might find out that the messages didn't arrive because they were automatically filtered and sent to a SPAM folder.

Types of Logging Data

In general, you can obtain two types of information from logging data:

- Operational data
  Most of Messaging Server logging is operational data such as:
  - The date and time a message entered the system
  - The sender and recipient of the message
  - When the message was written to disk
  - When the message was removed from disk and inserted into user's mailbox.
- Error conditions
  Data on error conditions comes from event logging. To obtain event logging data for error conditions, you need to pull together multiple items from different log files. You could then use a unique constant, such as message ID, to search and correlate the life cycle of a message as it passed from point to point through the system.

Types of Messaging Server Log Files

Messaging Server logging consists of three types of log files:

1. *MTA logs.* These logs provide operational data previously described for the Message Transfer Agent.
2. *Error logs.* These are the MTA debug logs, and the MTA subcomponent logs (that is, job controller, dispatcher and so on).
3. *Message Store and Service logs.* These logs provide messages from the http server, mshttpd, imap, and pop services, as well as the Admin service. The format of these logs differs from that of the first two types of logs.

The following table lists the different types of log files. By default, log files are located in the *msg-svr-base*`/data/log` directory. You can customize and view each type of log file individually.

Messaging Server Log Files

Type of Log File	Log File description	Default Name
Message Transfer Agent	Show information about message traffic through the MTA including date and time information, enqueue and dequeue information, and so on.	`mail.log_current`, `mail.log_yesterday`, `mail.log`
Connections	Contains remote machines (MTAs) that connect to this system to send email.	`connection.log`
Counters	Contains message trends in terms of messages sent and received on a per channel basis.	`counters`
Job Controller	Contains data on the master, job controller, sender, and dequeue channel programs.	`job_controller.log`
Dispatcher	Contains errors pertaining to the dispatcher. Turning on dispatcher debugging will increase the information.	`dispatcher.log`
Channel	Records errors pertaining to the channel. Keywords `master_debug` and `slave_debug` turn on channel debugging, which increases the verbosity of the channel log files. Level and type of information is controlled with the various `*_DEBUG` MTA options in `option.dat`.	*channel-name* `_master.log*` (example: `tcp_local_master.log*`) *channel-name* `_slave.log*` (example: `tcp_local_slave.log*`)
IMAP	Contains logged events related to IMAP4 activity of this server	`imap`, `imap.`*sequenceNum*`.`*timeStamp*
POP	Contains logged events related to POP3 activity of this server	`pop`, `pop.`*sequenceNum*`.`*timeStamp*
HTTP	Contains logged events related to HTTP activity of this server	`http`, `http.`*sequenceNum*`.`*timeStamp*
Default	Contains logged events related to other activity of this server, such as command-line utilities and other processes	`default`, `default.`*sequenceNum*`.`*timeStamp*
msgtrace	Contains trace information for the Message Store. File can grow very large very quickly. Monitor accordingly.	`msgtrace`
watcher	Monitors process failures and unresponsive services (see Services Monitored by watcher and msprobe) and will log error messages indicating specific failures.	`watcher`

where:

sequenceNum - Specifies an integer that specifies the order of creation of this log file compared to others in the log-file directory. Log files with higher sequence numbers are more recent than those with lower numbers. Sequence numbers do not roll over. They increase monotonically for the life of the server (beginning at server installation).

timeStamp - Specifies a large integer that specifies the date and time of file creation. (Its value is expressed in standard UNIX time: the number of seconds since midnight January 1, 1970.)

For example, a log file named `imap.63.915107696` would be the 63rd log file created in the directory of IMAP log files, created at 12:34:56 PM on December 31, 1998.

The combination of open-ended sequence numbering with a timestamp gives you more flexibility in

rotating, expiring, and selecting files for analyzing. For more specific suggestions, see Defining and Setting Service Logging Options.

Tools for Managing Logging

You can customize the policies for creating and managing Messaging Server log files by using the `configutil` command.

For the message store, the settings you specify affect which and how many events are logged. You can use those settings and other characteristics to refine searches for logged events when you are analyzing log files.

The MTA uses a separate logging facility. To configure MTA logging, you specify information in configuration files.

For log analysis and report generation beyond the capabilities of Messaging Server, you need to use other tools. You can manipulate log files on your own with text editors or standard system tools.

With a scriptable text editor supporting regular-expression parsing, you can potentially search for and extract log entries based on any of the criteria discussed in this information, and possibly sort the results or even generate sums or other statistics.

In UNIX environments you might also be able to modify and use existing report-generation tools that were developed to manipulate UNIX `syslog` files. If you wish to use a public-domain `syslog` manipulation tool, remember that you might need to modify it to account for the different date/time format and for the two extra components (*facility* and *logLevel*) that appear in Messaging Server log entries but not in `syslog` entries.

Tracking a Message Across the Various Log Files

The following describes how a message flows through the system, and at what point information gets written to the various log files. This description is meant to aid you in your understanding of how to use Message Server's log files to troubleshoot and resolve problems. See MTA Architecture to follow along.

1. A remote host makes a connection to the TCP socket on your Messaging Server host, requesting SMTP service.
2. The MTA dispatcher responds to the request, and hands off the connection to your messaging host's SMTP service.
   As the MTA is modular in design, it consists of a set of processes, including the job controller and the SMTP service dispatcher. The dispatcher takes the incoming TCP connection and sends it to the SMTP service. The SMTP service writes the message to disk to a channel area. The SMTP service understands the message's envelope parameters, such as sender and recipient. Configuration entries in the system tell what destination channel it belongs to.
3. The dispatcher writes to the `dispatcher.log` file that it forked a thread and made the thread available to incoming connection from a certain IP address.
4. The SMTP server writes to its `tcp_smtp_server.log` file, recording the dialog of what happens when the remote host connected to it and sent a message. This log file gets created when dispatcher hands off to SMTP server on the host's IP.
5. The SMTP server writes the message to a queue area on disk for a channel program such as `tcp_intranet`, and informs the job controller.
6. The job controller contacts the channel program.
7. The channel program delivers the message.
   Each channel has its own log file. However, these logs usually show the starting and stopping of the channel. To get more information, you need to enable debug level for the channel. However, as this can slow down your system and actually make problems more obscure if left on, you should only enable debug level when an actual problem is occurring.

> **Note**
> For efficiency, if a channel is already running for an existing process, and a new message comes in, the system does not spawn a new channel process. The currently running process picks up the new message.

8. The message is delivered to its next hop, which could be another host, another TCP connection, and so forth. This information is written to the `connection.log` file when `SEPARATE_CONNECTION_LOG` is enabled.
   At the same time that the SMTP server writes the message to a queue area on disk, the channel responsible for the message writes a record in the `mail.log_current` file. The record shows such information as the date and time the message was enqueued, the sender, the recipient, so forth. See MTA Message Logging Examples for more information. The most useful file for tracing the message is the `mail.log_current` file.

Managing MTA Message and Connection Logs

The MTA provides facilities for logging each message as it is enqueued and dequeued. It also provides dispatcher error and debugging output.

You can control logging on a per-channel basis or you can specify that message activity on all channels be logged. In the initial configuration, logging is disabled on all channels. For more information, see Enabling MTA Logging.

This section consists of the following subsections:

- Understanding the MTA Log Entry Format
- Enabling MTA Logging
- Specifying Additional MTA Logging Options
- MTA Message Logging Examples
- Enabling Dispatcher Debugging

When logging is enabled, the MTA writes an entry to the *msg-svr-base* `/data/log/mail.log_current` file each time a message passes through an MTA channel. Such log entries can be useful for gathering statistics on how many messages are passing through the MTA (or through particular channels). You can also use these log entries to investigate other issues, such as whether and when a message was sent or delivered.

The message return job, which runs every night around midnight, appends any existing `mail.log_yesterday` to the cumulative log file, `mail.log`, renames the current `mail.log_current` file to `mail.log_yesterday`, and then begins a new `mail.log_current` file. The message return job also performs the analogous operations for any `connection.log*` files.

While the MTA performs automatic rollovers to maintain the current file, you must manage the cumulative `mail.log` file by determining policies for tasks such as backing up the file, truncating the file, deleting the file, and so on.

When considering how to manage the log files, note that the MTA periodic return job will execute a site-supplied *msg-svr-base*`/data/site-programs/bin/daily_cleanup` script, if one exists. Thus some sites might choose to supply their own cleanup procedure that, for instance, renames the old `mail.log` file once a week (or once a month), and so on.

Note
With logging is enabled, the `mail.log` file steadily grows and, if left unchecked, consumes all available disk space. Monitor the size of this file and periodically delete unnecessary contents. You can also delete the entire file as another version will be created as needed.

Understanding the MTA Log Entry Format

The MTA log file is written as ASCII text. By default, each log file entry contains eight or nine fields as shown in the example below.

```
16-Feb-2007 14:54:13.72 tcp_local ims-ms EE 1 adam@sesta.com
rfc822;marlowe@siroe.com marlowe@ims-ms-daemon
```

The log entry shows:

1. The date and time the entry was made (in the example, 16-Feb-2007 14:54:13.72).
2. The channel name for the source channel (in the example, `tcp_local`).
3. The channel name for the destination channel (in the example, `ims-ms`). For SMTP channels, when `LOG_CONNECTION` is enabled, a plus (+) indicates inbound to the SMTP server; a minus (-) indicates outbound via the SMTP client.
4. The type of entry (in the example, `EE`). Entries can consist of a single action code (see Logging Entry Action Codes) or an action code and one or more modifier codes (see Logging Entry Modifier Codes). The format for entries is:

   ```
   <action_code><zero or more optional modifiers>
   ```

For example a logging entry code of `EEC` means that the email was Enqueued (action-code `E`) using ESMTP (modifier `E`) and SMTP Chunking (modifier `C`). Please refer to the tables below for details on the currently used action and modifier codes.

1. The size of the message (in the example, `1`). This is expressed in kilobytes by default, although this default can be changed by using the `BLOCK_SIZE` keyword in the MTA option file. The SMS channel can be configured to log a page count rather than file size in this field. See LOG_PAGE_COUNT.
2. The envelope `From:` address (in the example, `adam@sesta.com`). Note that for messages with an empty envelope `From:` address, such as notification messages, this field is blank.
3. The original form of the envelope `To:` address (in the example, `marlowe@siroe.com`).
4. The active (current) form of the envelope `To:` address (in the example, `marlowe@ims-ms-daemon`).
5. The delivery status (SMTP channels only).

The following three tables describe the logging entry codes.

Logging Entry Action Codes

Entry	Description
B	Bad command sent to the SMTP server. The recipient address field will contain the command that was rejected while the diagnostic field will contain the response the SMTP server gave. MTA channel option, MAX_B_ENTRIES, controls how many bad commands will be logged in a given session. Default is 10.
D	Successful dequeue
E	Successful enqueue
J	Rejection of attempted enqueue (rejection by slave channel program)
K	Recipient message rejected. If the sender requests `NOTIFY=NEVER` DSN flag set or if the message times out or if the message is manually returned (for example: `imsimta qm` "delete" command always generates a "K" record for each recipient, while a qm "return" command will generate a "K" record rather than an "R" record). This indicates that there was no notification sent to the sender per the sender's own request.This can be compared with "R" records, which are the same sort of rejection/time-out, but where a new notification message (back to the original sender) is also generated regarding this failed message.
Q	Temporary failure to dequeue
R	Recipient address rejected on attempted dequeue (rejection by master channel program), or generation of a failure/bounce message
S	LMTP deposit into the message store. This action code is used on the LMTP server side.
V	Warning message that will appear whenever a transaction is abnormally aborted. There will be one "V" record per enqueued recipient address.
W	Warning message sent to notify original sender that the message has not been delivered yet, but it is still in the queue being retried.
Z	Some successful recipients, but this recipient was temporarily unsuccessful; the original message file of all recipients was dequeued, and in its place a new message file for this and other unsuccessful recipients will be immediately enqueued

The following table describes the logging entry modifier codes.

Logging Entry Modifier Codes

Entry	Description
A	SASL authentication used.
B	SMTP BINARYMIME extension used (RFC 3030).
C	Chunking was used. Note that ESMTP has to be used for chunking to work, so you'll typically see field values like `EEC` or `DEC`.
E	An EHLO command was issued/accepted and therefore ESMTP was used.
L	LMTP was used.
Q	SMTP PIPELINING extension used (RFC 2920).
S	TLS/SSL used. S transaction log entries now increment the various submitted message counters associated with the channel.
U	BURL used (RFC 4468).

If `LOG_CONNECTION` is enabled (see LOG_CONNECTION), then an additional set of action codes will be

used. These are described below.

SMTP Channel's `LOG_CONNECTION` Action Codes + or - Entries

Entry	Description
C	Connection closed. A diagnostic field will follow. Written to connection.log_current (or mail.log_current if a single log file is being used). Used to record the reason why the connection was closed. In particular, if the connection was closed due to some session disconnect limit being reached, that fact will show up in the diagnostics field.
O	Connection opened
T	PORT_ACCESS log entry. Further details available in the PORT_ACCESS mapping table.
U	Logs SMTP authentication successes and failures. Format is the same as other O and C entries. In particular, the same application and transport information fields appear in same order. The username will be logged in the username field if it is known. Bit 7 (value 128) of the LOG_CONNECTION MTA option controls this.
X	Connection rejected
Y	Connection attempt failed before being established
I	ETRN command received

With `LOG_CONNECTION`, `LOG_FILENAME`, `LOG_MESSAGE_ID`, `LOG_NOTARY`, `LOG_PROCESS`, and `LOG_USERNAME` all enabled in the MTA Option file, the format becomes as shown in the example below. (The sample log entry line has been wrapped for typographic reasons; the actual log entry would appear on one physical line.)

```
16-Feb-2007 15:04:01.14 2bbe.5.3 tcp_local ims-ms
EE 1 service@siroe.com rfc822;adam@sesta.com
adam@ims-ms-daemon 20
/opt/SUNWmsgsr/data/queue/ims-ms/000/ZZf0r2i0HIaY1.01
&lt;0JDJ00803FAON200@mailstore.siroe.com> mailsrv
siroe.com (siroe.com [192.160.253.66])
```

Where the additional fields, beyond those already discussed above, are:

1. The process ID (expressed in hexadecimal), followed by a period (dot) character and a count. If this had been a multithreaded channel entry (that is, a `tcp_* channel` entry), there would also be a thread ID present between the process ID and the count. In the example, the process ID is 2bbe.5.3.
2. The `NOTARY` (delivery receipt request) flags for the message, expressed as an integer (in the example, 20).
3. The file name in the MTA queue area (in the example, `/opt/SUNWmsgsr/data/queue/ims-ms/000/ZZf0r2i0HIaY1.01`).
4. The message ID (in the example, <0JDJ00803FAON200@mailstore.siroe.com>).
5. The name of the executing process (in the example, `mailsrv`). On UNIX, for dispatcher processes such as the SMTP server, this will usually be `mailsrv` (unless SASL was used, in which case it will be the authenticated user name, for example, `*service@siroe.com`).
6. The connection information (in the example, `siroe.com (siroe.com [192.160.253.66])`. The connection information consists of the sending system or channel name, such as the name presented by the sending system on the HELO/EHLO line (for incoming SMTP messages), or the enqueuing channel's official host name (for other sorts of channels). In the case of TCP/IP channels, the sending system's real name, that is, the symbolic name as reported by a DNS

reverse lookup and/or the IP address, can also be reported within parentheses as controlled by the `ident*` channel keywords; see IDENT Lookups for an instance of the default `identnone` keyword, that selects display of both the name found from the DNS and IP address.

Enabling MTA Logging

To gather statistics for just a few particular MTA channels, enable the logging channel keyword on just those MTA channels of interest. Many sites prefer to enable logging on all MTA channels. In particular, if you are trying to track down problems, the first step in diagnosing some problems is to notice that messages are not going to the channel you expected or intended, and having logging enabled for all channels can help you investigate such problems.

To Enable MTA Logging on a Specific Channel

1. Edit the `imta.cnf` file.
   The file is located in the `/opt/SUNWmsgsr/config` directory.
2. To enable logging for a particular channel, add the `logging` keyword to the channel definition. For example:

   channel-name keyword1 keyword2 **logging**

   In addition, you can also set a number of configuration parameters such as directory path for log files, log levels, an so on. See Managing Message Store, Admin, and Default Service Logs.

   Note
   The message return job, which runs every night around midnight, appends any existing `mail.log_yesterday` to the cumulative log file, `mail.log`, renames the current `mail.log_current` file to `mail.log_yesterday`, and then begins a new `mail.log_current` file. It also performs the analogous operations for any `connection.log*` files. It is possible that `mail.log_yesterday` contains time stamps which have already passed over rotation time.

To Enable MTA Logging on All Channels

1. Edit the `imta.cnf` file.
   The file is located in the `/opt/SUNWmsgsr/config` directory.
2. Add the logging keyword to your `defaults` channel configuration file (see Configuring Channel Defaults). For example:

```
defaults notices 1 2 4 7 copywarnpost copysendpost postheadonly
noswitchchannel \
immnonurgent maxjobs 7 defaulthost siroe.com siroe.com logging

!
! delivery channel to local /var/mail store
l subdirs 20 viaaliasrequired maxjobs 7
mailhost.siroe.com
```

Specifying Additional MTA Logging Options

In addition to the basic information always provided when logging is enabled, you can specify that

additional, optional information fields be included by setting various LOG_* MTA options in the MTA Option file. The file specified with the IMTA_OPTION_FILE option in the IMTA tailor file (*msg-svr-base* /config/imta_tailor) specifies the MTA Option file. By default, this is the *msg-svr-base* /config/option.dat file.

> **Note**
> Messaging Server 7.0 and greater ignores the IMTA_OPTION_FILE parameter.

For complete details about the MTA Option file, see MTA Option file.

To Send MTA Logs to syslog

1. Edit the MTA Option file.
   0 is the default and means no syslog or event logging is performed.
2. Set the LOG_MESSAGES_SYSLOG option to a non-zero value to write MTA message log file entries to syslog.
   The absolute value of the non-zero value sets the syslog priority and facility mask. Negative values disable the generation of the regular mail.log* entries. Positive values mean that the syslog entries are generated in addition to the regular mail.log* entries.

Facility and priority numbers are located in the /usr/include/sys/syslog.h file.

To Control Formatting of Log Entries

1. Edit the MTA Option file.
2. Set the LOG_FORMAT option.
   - 1 (default) the standard format.
   - 2 requests non-null formatting: empty address fields are converted to the string "<>"
   - 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.
   - 4 causes log entries to be written in an XML-compatible format. Entry log entry appears as a single XML element containing multiple attributes and no sub-elements. Three elements are currently defined, en for enqueue/dequeue entries, co for connection entries, and he for header entries.
     Enqueue/dequeue (en) elements can have the following attributes:

```
ts - time stamp (always present)
no - node name (present if LOG_NODE=1)
pi - process id (present if LOG_PROCESS=1)
sc - source channel (always present)
dc - destination channel (always present)
ac - action (always present)
sz - size (always present)
so - source address (always present)
od - original destination address (always present)
de - destination address (always present)
rf - recipient flags (present if LOG_NOTARY=1)
fi - filename (present if LOG_FILENAME=1)
ei - envelope id (present if LOG_ENVELOPE_ID=1)
mi - message id (present if LOG_MESSAGE_ID=1)
us - username (present if LOG_USERNAME=1)
ss - source system (present if bit 0 of LOG_CONNECTION
is set and source system information is available)
se - sensitivity (present if LOG_SENSITIVITY=1)
pr - priority (present if LOG_PRIORITY=1)
in - intermediate address (present if LOG_INTERMEDIATE=1)
ia - initial address (present if bit 0 of LOG_INTERMEDIATE
is set and intermediate address information is available)
fl - filter (present if LOG_FILTER=1 and filter information
is available)
re - reason (present if LOG_REASON=1 and reason string is set)
di - diagnostic (present if diagnostic info available)
tr - transport information (present if bit 5 of LOG_CONNECTION
is set and transport information is available)
ap - application information (present if bit 6 of LOG_CONNECTION
is set and application information is available)
qt - the number of seconds the message has spent in the queue
(LOG_QUEUE_TIME=1)
```

Here is a sample en entry:

```
<en ts="2004-12-08T00:40:26.70" pi="0d3730.10.43"
sc="tcp_local"
dc="l" ac="E" sz="12"
so="info-E8944AE8D033CB92C2241E@whittlesong.com"
od="rfc822;ned+2Bcharsets@mauve.sun.com"
de="ned+charsets@mauve.sun.com" rf="22"
fi="/path/ZZ01LI4XPX0DTM00IKA8.00"
ei="01LI4XPQR2EU00IKA8@mauve.sun.com"
mi="&lt;11a3b401c4dd01$7c1c1ee0$1906fad0@elara>" us=""
ss="elara.whittlesong.com ([208.250.6.25])"
in="ned+charsets@mauve.sun.com"
ia="ietf-charsets@innosoft.com"
fl="spamfilter1:rvLiXh158xWdQKa9iJ0d7Q==, addheader, keep"/>
```

Note that this entry has been wrapped for clarity; actual log file entries always appear on a single line.
Connection (co) entries can have the following attributes:

```
ts - time stamp (always present, also used in en entries)
no - node name (present if LOG_NODE=1, also used in en entries)
pi - process id (present if LOG_PROCESS=1, also used in en
entries)
sc - source channel (always present, also used in en entries)
dr - direction (always present)
ac - action (always present, also used in en entries)
tr - transport information (always present, also used in en
entries)
ap - application information (always present, also used in en
entries)
mi - message id (present only if message id info available,
also used in en entries)
us - username (present only if username information available,
also
used in en entries)

di - diagnostic (present only if diagnostic information available,
also used in en entries)
ct - the length of the connection, in seconds. (LOG_QUEUE_TIME=1,
also used in en entries)
```

Here is a sample co entry:

```
<co ts="2004-12-08T00:38:28.41" pi="1074b3.61.281"
sc="tcp_local" dr="+"
ac="O" tr="TCP|209.55.107.55|25|209.55.107.104|33469"
ap="SMTP"/>
```

Header (he) entries have the following attributes:

```
ts - time stamp (always present, also used in en entries)
no - node name (present if LOG_NODE=1, also used in en entries)
pi - process id (present if LOG_PROCESS=1, also used in en
entries)
va - header line value (always present)
```

Here is a sample he entry:

```
<he ts="2004-12-08T00:38:31.41" pi="1074b3.61.281"
va="Subject: foo"/>
```

To Correlate Log Message Entries

1. Edit the MTA Option file.
2. Set the `LOG_MESSAGE_ID` option to 1.
   A value of 0 is the default and indicates that message IDs are not saved in the `mail.log*` files.

To Log Amount of Time Messages Have Spent in the Queue

1. Edit the MTA Option file.
2. Set the LOG_QUEUE_TIME option to 1.
   This option logs the amount of time messages spent in the queue. The queue time is logged as an integer value in seconds. It appears immediately after the application information string in non-XML format logs. The attribute name in XML formatted logs for this value is qt.

To Identify Message Delivery Retries

1. Edit the MTA Option file.
2. Set the LOG_FILENAME option to 1.
   This option makes it easier to immediately spot how many times the delivery of a particular message file has been retried. This option can also be useful in understanding when the MTA does or does not split a message to multiple recipients into separate message file copies on disk.

To Log TCP/IP Connections

1. Edit the MTA Option file.
2. Set the LOG_CONNECTION option.
   This option causes the MTA to log TCP/IP connections, as well as message traffic. The connection log entries are written to the mail.log* files by default. Optionally, the connection log entries can be written to connection.log* files. See the SEPARATE_CONNECTION_LOG option for more information.

To Write Entries to the connection.log File

1. Edit the MTA Option file.
2. Set the SEPARATE_CONNECTION_LOG option to 1.
   Use this option to specify that connection log entries instead be written to connection.log files. The default value of 0 causes the connection logging to be stored in the MTA log files.

To Correlate Log Messages by Process ID

1. Edit the MTA Option file.
2. Set the LOG_PROCESS option.
   When used in conjunction with LOG_CONNECTION, this option enables correlation by process ID of which connection entries correspond to which message entries.

To Save User Names Associated with a Process That Enqueues Mail to the mail.log File

1. Edit the MTA Option file.
2. Set the LOG_USERNAME option.
   This option controls whether or not the user name associated with a process that enqueues mail is saved in the mail.log file. For SMTP submissions where SASL (SMTP AUTH) is used, the user name field will be the authenticated user name (prefixed with an asterisk character).

MTA Message Logging Examples

The exact field format and list of fields logged in the MTA message files vary according to the logging options set. This section shows a few examples of interpreting typical sorts of log entries.

For a description of additional, optional fields, see Specifying Additional MTA Logging Options.

> **Note**
> For typographic reasons, log file entries will be shown folded onto multiple lines. Actual log file entries are one line per entry.

When reviewing a log file, keep in mind that on a typical system many messages are being handled at once. Typically, the entries relating to a particular message will be interspersed among entries relating to other messages being processed during that same time. The basic logging information is suitable for gathering a sense of the overall numbers of messages moving through the MTA.

If you wish to correlate particular entries relating to the same message to the same recipient(s), enable `LOG_MESSAGE_ID`. To correlate particular messages with particular files in the MTA queue area, or to see from the entries how many times a particular not-yet-successfully-dequeued message has had delivery attempted, enable `LOG_FILENAME`. For SMTP messages (handled via a TCP/IP channel), if you want to correlate TCP connections to and from remote systems with the messages sent, enable `LOG_PROCESS` and some level of `LOG_CONNECTION`.

MTA Logging Example: User Sends an Outgoing Message

The example below shows a fairly basic example of the sort of log entries one might see if a local user sends a message out an outgoing TCP/IP channel, for example, to the Internet. In this example, `LOG_CONNECTION` is enabled. The lines marked with (1) and (2) are one entry---they would appear on one physical line in an actual log file. Similarly, the lines marked with (3) - (7) are one entry and would appear on one physical line.

Example MTA Logging: A Local User Sends An Outgoing Message

```
16-Feb-2007 15:41:32.36 tcp_intranet tcp_local EE 1 (1)
adam@sesta.com rfc822;marlowe@siroe.com marlowe@siroe.com (2)
siroe.com (siroe.com [192.160.253.66])

16-Feb-2007 15:41:34.73 tcp_local DE 1 (3)
adam@sesta.com rfc822;marlowe@siroe.com marlowe@siroe.com (4)
thor.siroe.com dns;thor.siroe.com

(TCP|206.184.139.12|2788|192.160.253.66|25) (5)

(thor.siroe.com ESMTP Sendmail ready Thu 15 Feb 2007 21:37:29 -0700
[MST]) (6)

smtp;250 2.1.5 &lt;marlowe@siroe.com>... Receipt ok (7)
```

1. This line shows the date and time of an enqueue with ESMTP (EE) from the `tcp_intranet` channel to the `tcp_local` channel of a one (1) block message.
2. This is part of the same physical line of the log file as (1), presented here as a separate line for typographical convenience. It shows the envelope `From:` address, in this case adam@sesta.com, and the original version and current version of the envelope `To:` address, in this case `marlowe@siroe.com`.
3. This shows the date and time of a dequeue with ESMTP (DE) from the `tcp_local` channel of a one (1) block message that is, a successful send by the `tcp_local` channel to some remote SMTP server.
4. This shows the envelope `From:` address, the original envelope `To:` address, and the current form of the envelope `To:` address.
5. This shows that the actual system to which the connection was made is named `thor.siroe.com` in the DNS, that the local sending system has IP address 206.184.139.12 and is sending from port 2788, that the remote destination system has IP address 192.160.253.66 and the connection port on the remote destination system is port 25.
6. This shows the SMTP banner line of the remote SMTP server.
7. This shows the SMTP status code returned for this address; 250 is the basic SMTP success code and in addition, this remote SMTP server responds with extended SMTP status codes and some additional text.

MTA Logging Example: Including Optional Logging Fields

This example shows a logging entry similar to that shown in Example with `LOG_FILENAME=1` and `LOG_MESSAGE_ID=1` showing the file name (1 and 3 below) and message ID (2 and 4 below). The message ID in particular can be used to correlate which entries relate to which message.

Example MTA Logging: Including Optional Logging Fields

```
16-Feb-2007 15:41:32.36 tcp_intranet tcp_local EE 1
adam@sesta.com rfc822;marlowe@siroe.com marlowe@siroe.com
/opt/SUNWmsgsr/data/queue/tcp_local/002/ZZf0r4i0Wdy51.01 (1)
&lt;0JDJ00D02IBWDX00@sesta.com> (2)
siroe.com (siroe.com [192.160.253.66])

16-Feb-2007 15:41:34.73 tcp_local DE 1
adam@sesta.com rfc822;marlowe@siroe.com marlowe@siroe.com
/opt/SUNWmsgsr/data/queue/tcp_local/002/ZZf0r4i0Wdy51.01 (3)
&lt;0JDJ00D02IBWDX00@sesta.com> (4)
thor.siroe.com dns;thor.siroe.com
(TCP|206.184.139.12|2788|192.160.253.66|25)
(thor.siroe.com ESMTP Sendmail ready at Thu, 15 Feb 2007 21:37:29 -0700
[MST])
smtp;250 2.1.5 &lt;marlowe@siroe.com>... Recipient ok
```

MTA Logging Example: Sending to a List

This example illustrates sending to multiple recipients with `LOG_FILENAME=1`, `LOG_MESSAGE_ID=1`, and `LOG_CONNECTION=1` enabled. Here user `adam@sesta.com` has sent to the MTA mailing list `test-list@sesta.com`, which expanded to `bob@sesta.com`, `carol@varrius.com`, and `david@varrius.com`. Note that the original envelope `To:` address is `test-list@sesta.com` for each recipient, though the current envelope `To:` address is each respective address. Note how the message ID is the same throughout, though two separate files (one for the `l` channel and one going out the `tcp_local` channel) are involved.

Example MTA Logging: Sending to a List

```
20-Feb-2007 14:00:16.46 tcp_local tcp_local EE 1
adam@sesta.com rfc822;test-list@sesta.com carol@varrius.com
/opt/SUNWmsgsr/data/queue/tcp_local/004/ZZf0r2D0yuej4.01
&lt;0JDQ00706R0FX100@sesta.com>
siroe.com (siroe.com [192.160.253.66])

20-Feb-2007 14:00:16.47 tcp_local tcp_local EE 1
adam@sesta.com rfc822;test-list@sesta.com david@varrius.com
/opt/SUNWmsgsr/data/queue/tcp_local/004/ZZf0r2D0yuej4.01
&lt;0JDQ00706R0FX100@sesta.com>
siroe.com (siroe.com [192.160.253.66])

20-Feb-2007 14:00:16.48 tcp_local ims-ms EE 1
adam@sesta.com rfc822;test-list@sesta.com bob@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/008/ZZf0r2D0yuej6.01
&lt;0JDQ00706R0FX100@sesta.com>
siroe.com (siroe.com [192.160.253.66])

20-Feb-2007 14:00:16.68 ims-ms D 1
adam@sesta.com rfc822;test-list@sesta.com bob@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/008/ZZf0r2D0yuej6.01
&lt;0JDQ00706R0FX100@sesta.com>

20-Feb-2007 14:00:17.73 tcp_local DE 1
adam@sesta.com rfc822;test-list@sesta.com carol@varrius.com
/opt/SUNWmsgsr/data/queue/tcp_local/004/ZZf0r2D0yuej4.01
&lt;0JDQ00706R0FX100@sesta.com>
gw.varrius.com dns;gw.varrius.com
(TCP|206.184.139.12|2788|192.160.253.66|25)
(gw.varrius.com -- SMTP Sendmail)
smtp;250 2.1.5 &lt;carol@varrius.com >... Recipient ok

20-Feb-2007 14:00:17.75 tcp_local DE 1
adam@sesta.com rfc822;test-list@sesta.com david@varrius.com
/opt/SUNWmsgsr/data/queue/tcp_local/004/ZZf0r2D0yuej4.01
&lt;0JDQ00706R0FX100@sesta.com>
gw.varrius.com dns;gw.varrius.com
(TCP|206.184.139.12|2788|192.160.253.66|25)
(gw.varrius.com -- SMTP Sendmail)
smtp;250 2.1.5 &lt;david@varrius.com>... Recipient ok
```

MTA Logging Example: Sending to a Nonexistent Domain

This example illustrates an attempt to send to a nonexistent domain (here `very.bogus.com`); that is, sending to a domain name that is not noticed as nonexistent by the MTA's rewrite rules and that the MTA matches to an outgoing TCP/IP channel. This example assumes the MTA option settings of `LOG_FILENAME=1` and `LOG_MESSAGE_ID=1`.

When the TCP/IP channel runs and checks for the domain name in the DNS, the DNS returns an error that no such name exists. Note the "rejection" entry (R), as seen in (5), with the DNS returning an error that this is not a legal domain name, as seen in (6).

Because the address is rejected after the message has been submitted, the MTA generates a bounce

message to the original sender. The MTA enqueues the new rejection message to the original sender (1), and sends a copy to the postmaster (4) before deleting the original outbound message (the R entry shown in (5)).

Notification messages, such as bounce messages, have an empty envelope From: address--~~as seen, for instance, in (2) and (8)~~ --in which the envelope From: field is shown as an empty space. The initial enqueue of a bounce message generated by the MTA shows the message ID for the new notification message followed by the message ID for the original message (3). (Such information is not always available to the MTA, but when it is available to be logged, it allows correlation of the log entries corresponding to the outbound failed message with the log entries corresponding to the resulting notification message.) Such notification messages are enqueued to the process channel, which in turn enqueues them to an appropriate destination channel (7).

Example MTA Logging: Sending to a Nonexistent Domain

```
20-Feb-2007 14:17:07.77 tcp_intranet tcp_local E 1
adam@sesta.com rfc822;user@very.bogus.com user@very.bogus.com
/opt/SUNWmsgsr/data/queue/tcp_local/008/ZZf0r2D0CVaL0.00
&lt;0JDQ00903RS89T00@sesta.com>
siroe.com (siroe.com [192.160.253.66])

20-Feb-2007 14:17:08.24 tcp_local process E 1 (1)
rfc822;adam@sesta.com adam@sesta.com (2)
/opt/SUNWmsgsr/data/queue/process/ZZf0r2D0CVbR0.00
&lt;0JDQ00904RSK9Z00@sesta.com>,&lt;0JDQ00903RS89T00@sesta.com> (3)
tcp-daemon.mailhost.sesta.com

20-Feb-2007 14:17:08.46 tcp_local process E 1 (4)
rfc822;postmaster@sesta.com postmaster@sesta.com
/opt/SUNWmsgsr/data/queue/process/ZZf0r2D0CVbR1.00
&lt;0JDQ00906RSK9Z00@sesta.com>,&lt;0JDQ00903RS89T00@sesta.com>
tcp-daemon.mailhost.sesta.com

20-Feb-2007 14:17:08.46 tcp_local R 1 (5)
adam@sesta.com rfc822;user@very.bogus.com user@very.bogus.com
/opt/SUNWmsgsr/data/queue/tcp_local/008/ZZf0r2D0CVaL0.00
&lt;0JDQ00903RS89T00@sesta.com>
Illegal host/domain name found (6)
(TCP active open: Failed gethostbyname() on very.bogus.com, resolver
errno = 1)

20-Feb-2007 14:17:09.21 process ims-ms E 3 (7)
rfc822;adam@sesta.com adam@ims-ms-daemon (8)
/opt/SUNWmsgsr/data/queue/ims-ms/018/ZZf0r2D0CVbS1.00
&lt;0JDQ00904RSK9Z00@sesta.com>
process-daemon.mailhost.sesta.com

20-Feb-2007 14:17:09.72 process ims-ms E 3
rfc822;postmaster@sesta.com postmaster@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/014/ZZf0r2D0CVbS2.00
&lt;0JDQ00906RSK9Z00@sesta.com>
process-daemon.mailhost.sesta.com

20-Feb-2007 14:17:09.73 ims-ms D 3
rfc822;adam@sesta.com adam@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/018/ZZf0r2D0CVbS1.00
&lt;0JDQ00904RSK9Z00@sesta.com>

20-Feb-2007 14:17:09.84 ims-ms D 3
rfc822;postmaster@sesta.com postmaster@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/014/ZZf0r2D0CVbS2.00
&lt;0JDQ00906RSK9Z00@sesta.com>
```

MTA Logging Example: Sending to a Nonexistent Remote User

This example illustrates an attempt to send to a bad address on a remote system. This example assumes MTA option settings of `LOG_FILENAME=1` and `LOG_MESSAGE_ID=1`, and channel option settings of `LOG_BANNER=1` and `LOG_TRANSPORTINFO=1`. Note the rejection entry (R), seen in (1). But

in contrast to the rejection entry in Example, note that the rejection entry here shows that a connection to a remote system was made, and shows the SMTP error code issued by the remote SMTP server, (2) and (3). The inclusion of the information shown in (2) is due to setting the channel options `LOG_BANNER=1` and `LOG_TRANSPORTINFO=1`.

Example MTA Logging: Sending to a Nonexistent Remote User

```
26-Feb-2007 13:56:35.16 tcp_intranet tcp_local EE 1
adam@sesta.com rfc822;nonesuch@siroe.com nonesuch@siroe.com
/opt/SUNWmsgsr/data/queue/tcp_local/000/ZZf0s690a3mf2.01
&lt;0JE100J08UU24H00@sesta.com>
siroe.com (siroe.com [192.160.253.66])

26-Feb-2007 13:56:35.19 tcp_local process E 1
rfc822;adam@sesta.com adam@sesta.com
/opt/SUNWmsgsr/data/queue/process/ZZf0s690a3ml2.00
&lt;0JE100J09UUB4N00@sesta.com>,&lt;0JE100J08UU24H00@sesta.com>
tcp-daemon.mailhost.sesta.com

26-Feb-2007 13:56:35.20 tcp_local process E 1
rfc822;postmaster@sesta.com postmaster@sesta.com
/opt/SUNWmsgsr/data/queue/process/ZZf0s690a3ml3.00
&lt;0JE100J0BUUB4N00@sesta.com>,&lt;0JE100J08UU24H00@sesta.com>
tcp-daemon.mailhost.sesta.com

26-Feb-2007 13:56:35.20 tcp_local RE 1 (1)
adam@sesta.com rfc822;nonesuch@siroe.com nonesuch@siroe.com

/opt/SUNWmsgsr/data/queue/tcp_local/000/ZZf0s690a3mf2.01
&lt;0JE100J08UU24H00@sesta.com>
thor.siroe.com dns;thor.siroe.com
(TCP|206.184.139.12|2788|192.160.253.66|25) (2)
(thor.siroe.com -- Server ESMTP [Sun Java System Messaging
Server 6.2-8.01 [built Feb 16 2007]])
smtp;550 5.1.1 unknown or illegal alias: nonesuch@siroe.com (3)

26-Feb-2007 13:56:35.62 process ims-ms E 4
rfc822;adam@sesta.com adam@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/003/ZZf0s690a3mm5.00
&lt;0JE100J09UUB4N00@sesta.com>
process-daemon.mailhost.sesta.com

26-Feb-2007 13:56:36.07 process ims-ms E 4
rfc822;postmaster@sesta.com postmaster@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/016/ZZf0s690a3nm7.01
&lt;0JE100J0BUUB4N00@sesta.com>
process-daemon.mailhost.sesta.com

26-Feb-2007 13:56:35.83 ims-ms D 4
rfc822;adam@sesta.com adam@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/003/ZZf0s690a3mm5.00
&lt;0JE100J09UUB4N00@sesta.com>

26-Feb-2007 13:56:36.08 ims-ms D 4
rfc822;postmaster@sesta.com postmaster@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/016/ZZf0s690a3nm7.01
&lt;0JE100J0BUUB4N00@sesta.com>
```

MTA Logging Example: Rejecting a Remote Side's Attempt to Submit a Message

This example illustrates the sort of log file entry resulting when the MTA rejects a remote side's attempt

to submit a message. (This example assumes that no optional LOG_* options are enabled, so only the basic fields are logged in the entry. Note that enabling the LOG_CONNECTION option, in particular, would result in additional informative fields in such J entries.) In this case, the example is for an MTA that has set up SMTP relay blocking (see Configuring SMTP Relay Blocking) with an ORIG_SEND_ACCESS mapping, including:

```
ORIG_SEND_ACCESS

! ...numerous entries omitted...
!
tcp_local|*|tcp_local|* $NRelaying$ not$ permitted
```

and where alan@very.bogus.com is not an internal address. Hence the attempt of the remote user harold@varrius.com to relay through the MTA system to the remote user alan@very.bogus.com is rejected.

Example MTA Logging: Rejecting a Remote Side's Attempt to Submit a Message

```
26-Feb-2007 14:10:06.89 tcp_local JE 0 (1)
harold@varrius.com rfc822; alan@very.bogus.com (2)
530 5.7.1 Relaying not allowed: alan@very.bogus.com (3)
```

1. This log shows the date and time the MTA rejects a remote side's attempt to submit a message. The rejection is indicated by a J record. (Cases where an MTA channel is attempting to send a message which is rejected is indicated by R records, as shown in Example and Example).

   Note
   The last J record written to the log will have an indication stating that it is the last for a given session. Also, the current version of Messaging Server does not place a limit on the number of J records.

2. The attempted envelope From: and To: addresses are shown. In this case, no original envelope To: information was available so that field is empty.
3. The entry includes the SMTP error message the MTA issued to the remote (attempted sender) side.

MTA Logging Example: Multiple Delivery Attempts

This example illustrates the sort of log file entries resulting when a message cannot be delivered upon the first attempt, so the MTA attempts to send the message several times. This example assumes option settings of LOG_FILENAME=1 and LOG_MESSAGE_ID=1.

Example MTA Logging: Multiple Delivery Attempts

```
26-Feb-2007 14:38:16.27 tcp_intranet tcp_local EE 1 (1)
adam@sesta.com rfc822;user@some.org user@some.org
/opt/SUNWmsgsr/data/queue/tcp_local/001/ZZf0s690kN_y0.00
&lt;0JE100L05WRJIC00@sesta.com>

26-Feb-2007 14:38:16.70 tcp_local Q 1 (2)
adam@sesta.com rfc822;user@some.org user@some.org
/opt/SUNWmsgsr/data/queue/tcp_local/001/ZZf0s690kN_y0.00 (3)
&lt;0JE100L05WRJIC00@sesta.com>
TCP active open: Failed connect() 192.1.1.1:25 Error: no route to host
(4)

...several hours worth of entries...

26-Feb-2007 16:58:11.20 tcp_local Q 1 (5)
adam@sesta.com rfc822;user@some.org user@some.org
/opt/SUNWmsgsr/data/queue/tcp_local/001/ZYf0s690kN_y0.01 (6)
&lt;0JE100L05WRJIC00@sesta.com>
TCP active open: Failed connect() 192.1.1.1:25 Error: no route to host

...several hours worth of entries...

26-Feb-2007 19:15:12.11 tcp_local Q 1
adam@sesta.com rfc822;user@some.org user@some.org
/opt/SUNWmsgsr/data/queue/tcp_local/001/ZXf0s690kN_y0.00 (7)
&lt;0JE100L05WRJIC00@sesta.com>
TCP active open: Failed connect() 192.1.1.1:25 Error: Connection refused
(8)

...several hours worth of entries...

26-Feb-2007 22:41:12.63 tcp_local DE 1 (9)
adam@sesta.com rfc822;user@some.org user@some.org
/opt/SUNWmsgsr/data/queue/tcp_local/001/ZXf0s690kN_y0.00
&lt;0JE100L05WRJIC00@sesta.com>
host.some.org dns;host.some.org (TCP|206.184.139.12|2788|192.1.1.1|25)
(All set, fire away)
smtp;250 2.1.5 &lt;user@some.org >... Recipient ok
```

1. The message comes in the `tcp_internal` channel---perhaps from a POP or IMAP client, or perhaps from another host within the organization using the MTA as an SMTP relay; the MTA enqueues it to the outgoing `tcp_local` channel.
2. The first delivery attempt fails, as indicated by the Q entry.
3. That this is a first delivery attempt can be seen from the ZZ* filename.
4. This delivery attempt failed when the TCP/IP package could not find a route to the remote side. As opposed to Example, the DNS did not object to the destination domain name, `some.org`; rather, the "no route to host" error indicates that there is some network problem between the sending and receiving side.
5. The next time the MTA periodic job runs it reattempts delivery, again unsuccessfully.
6. The file name is now ZY*, indicating that this is a second attempt.
7. The file name is ZX* for this third unsuccessful attempt.
8. The next time the periodic job reattempts delivery the delivery fails, though this time the TCP/IP package is not complaining that it cannot get through to the remote SMTP server, but rather the remote SMTP server is not accepting connections. (Perhaps the remote side fixed their network problem, but has not yet brought their SMTP server back up---or their SMTP server is swamped

handling other messages and hence was not accepting connections at the moment the MTA tried to connect.)
9. Finally the message is dequeued.

MTA Logging Example: Incoming SMTP Message Routed Through the Conversion Channel

This example illustrates the case of a message routed through the conversion channel. The site is assumed to have a CONVERSIONS mapping table such as:

```
CONVERSIONS

IN-CHAN=tcp_local;OUT-CHAN=ims-ms;CONVERT Yes
```

This example assumes option settings of `LOG_FILENAME=1` and `LOG_MESSAGE_ID=1`.

Example MTA Logging: Incoming SMTP Message Routed Through the Conversion Channel

```
26-Feb-2007 15:31:04.17 tcp_local conversion EE 1 (1)
amy@siroe.edu rfc822;bert@sesta.com bert@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/conversion/ZZf0s090wFwx2.01
&lt;0JE100206Z7J5F00@siroe.edu>

26-Feb-2007 15:31:04.73 conversion ims-ms E 1 (2)
amy@siroe.edu rfc822;bert@sesta.com bert@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/007/ZZf0s090wMwq1.00
&lt;0JE100206Z7J5F00@siroe.edu>

26-Feb-2007 15:31:04.73 conversion D 1 (3)
amy@siroe.edu rfc822;bert@sesta.com bert@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/conversion/ZZf0s090wFwx2.01
&lt;0JE100206Z7J5F00@siroe.edu>

26-Feb-2007 15:31:04.73 ims-ms D 1 (4)
amy@siroe.edu rfc822;bert@sesta.com bert@ims-ms-daemon
/opt/SUNWmsgsr/data/queue/ims-ms/007/ZZf0s090wMwq1.00
&lt;0JE100206Z7J5F00@siroe.edu>
```

1. The message from external user amy@siroe.edu comes in addressed to the `ims-ms` channel recipient bert@sesta.com. The `CONVERSIONS` mapping entry, however, causes the message to be initially enqueued to the conversion channel (rather than directly to the `ims-ms` channel).
2. The conversion channel runs and enqueues the message to the `ims-ms` channel.
3. Then the conversion channel can dequeue the message (delete the old message file).
4. And finally the `ims-ms` channel dequeues (delivers) the message.

MTA Logging Example: Outbound Connection Logging

This example illustrates log output for an outgoing message when connection logging is enabled, via `LOG_CONNECTION=3`. `LOG_PROCESS=1`, `LOG_MESSAGE_ID=1` and `LOG_FILENAME=1` are also assumed in this example. The example shows the case of user `adam@sesta.com` sending the same message (note that the message ID is the same for each message copy) to three recipients, `bobby@hosta.sesta.com`, `carl@hosta.sesta.com`, and `dave@hostb.sesta.com`. This example assumes that the message is going out a `tcp_local` channel marked (as such channels usually are) with the `single_sys` channel keyword. Therefore, a separate message file on disk will be created for each set of recipients to a separate host name, as seen in (1), (2), and (3), where the

bobby@hosta.sesta.com and carl@hosta.sesta.com recipients are stored in the same message file, but the dave@hostb.sesta.com recipient is stored in a different message file.

Example MTA Logging: Outbound Connection Logging

```
28-Feb-2007 09:13:19.18 409f.3.1 tcp_intranet tcp_local EE 1
adam@sesta.com rfc822;bobby@hosta.sesta.com bobby@hosta.sesta.com
/opt/SUNWmsgsr/data/queue/tcp_local/000/ZZf0s4g0G2Zt0.00 (1)
&lt;0JE500C0371HRJ00@sesta.com>
siroe.com (siroe.com [192.160.253.66])

28-Feb-2007 09:13:19.18 409f.3.1 tcp_intranet tcp_local EE 1
adam@sesta.com rfc822;carl@hosta.sesta.com carl@hosta.sesta.com
/opt/SUNWmsgsr/data/queue/tcp_local/000/ZZf0s4g0G2Zt0.00 (2)
&lt;0JE500C0371HRJ00@sesta.com>
siroe.com (siroe.com [192.160.253.66])

28-Feb-2007 09:13:19.19 409f.3.2 tcp_intranet tcp_local EE 1
adam@sesta.com rfc822;dave@hostb.sesta.com dave@hostb.sesta.com
/opt/SUNWmsgsr/data/queue/tcp_local/004/ZZf0s4g0G2Zt1.00 (3)
&lt;0JE500C0371HRJ00@sesta.com>
siroe.com (siroe.com [192.160.253.66])

28-Feb-2007 09:13:19.87 40a5.2.0 tcp_local - O (4)
TCP|206.184.139.12|5900|206.184.139.66|25
SMTP/hostb.sesta.com/mailhub.sesta.com (5)

28-Feb-2007 09:13:20.23 40a5.3.4 tcp_local - O (6)
TCP|206.184.139.12|5901|206.184.139.70|25
SMTP/hosta.sesta.com/hosta.sesta.com (7)

28-Feb-2007 09:13:20.50 40a5.2.5 tcp_local DE 1
adam@sesta.com rfc822;bobby@hosta.sesta.com bobby@hosta.sesta.com
/opt/SUNWmsgsr/data/queue/tcp_local/000/ZZf0s4g0G2Zt0.00
&lt;0JE500C0371HRJ00@sesta.com>
hosta.sesta.com dns;hosta.sesta.com (8)
(TCP|206.184.139.12|5901|206.184.139.70|25)
(hosta.sesta.com -- Server ESMTP [Sun Java System Messaging Server
6.2-8.01 [built Feb 16 2007]])
smtp;250 2.1.5 bobby@hosta.sesta.com and options OK.

28-Feb-2007 09:13:20.50 40a5.2.5 tcp_local DE 1
adam@sesta.com rfc822;carl@hosta.sesta.com carl@hosta.sesta.com
/opt/SUNWmsgsr/data/queue/tcp_local/000/ZZf0s4g0G2Zt0.00
&lt;0JE500C0371HRJ00@sesta.com>
hosta.sesta.com dns;hosta.sesta.com
(TCP|206.184.139.12|5901|206.184.139.70|25)
(hosta.sesta.com -- Server ESMTP [Sun Java System Messaging Server
6.2-8.01 [built Feb 16 2007]])
smtp;250 2.1.5 carl@hosta.sesta.com and options OK.

28-Feb-2007 09:13:20.50 40a5.2.6 tcp_local - C (9)
TCP|206.184.139.12|5901|206.184.139.70|25
SMTP/hosta.sesta.com/hosta.sesta.com
```

```
28-Feb-2007 09:13:21.13 40a5.3.7 tcp_local DE 1
adam@sesta.com rfc822;dave@hostb.sesta.com dave@hostb.sesta.com
/opt/SUNWmsgsr/data/queue/tcp_local/004/ZZf0s4g0G2Zt1.00
&lt;0JE500C0371HRJ00@sesta.com>
mailhub.sesta.com dns;mailhub.sesta.com
(TCP|206.184.139.12|5900|206.184.139.66|25)
(mailhub.sesta.com ESMTP Sendmail ready at Tue, 27 Feb 2007 22:19:40
GMT)
smtp;250 2.1.5 &lt;dave@hostb.sesta.com>... Recipient ok

28-Feb-2007 09:13:21.33 40a5.3.8 tcp_local - C (10)
```

```
TCP|206.184.139.12|5900|206.184.139.66|25
SMTP/hostb.sesta.com/mailhub.sesta.com
```

1. The message is enqueued to the first recipient...
2. ....and to the second recipient...
3. ....and to the third recipient.
4. Having `LOG_CONNECTION=3` set causes the MTA to write this entry. The minus, `-`, indicates that this entry refers to an outgoing connection. The `0` means that this entry corresponds to the opening of the connection. Also note that the process ID here is the same, 40a5, since the same process is used for the multithreaded TCP/IP channel for these separate connection opens, though this open is being performed by thread 2 vs. thread 3.
5. As there are two separate remote systems to which to connect, the multithreaded SMTP client in separate threads opens up a connection to each---the first in this entry, and the second shown in 7. This part of the entry shows the sending and destination IP numbers and port numbers, and shows both the initial host name, and the host name found by doing a DNS lookup. In the `SMTP/`*initial-host*`/`*dns-host* clauses, note the display of both the initial host name, and that used after performing a DNS MX record lookup on the initial host name: `mailhub.sesta.com` is apparently an MX server for `hostb.sesta.com`.
6. The multithreaded SMTP client opens up a connection to the second system in a separate thread (though the same process).
7. As there are two separate remote systems to which to connect, the multithreaded SMTP client in separate threads opens up a connection to each---the second in this entry, and the first shown above in 5. This part of the entry shows the sending and destination IP numbers and port numbers, and shows both the initial host name, and the host name found by doing a DNS lookup. In this example, the system `hosta.sesta.com` apparently receives mail directly itself.
8. Besides resulting in specific connection entries, `LOG_CONNECTION=3` also causes inclusion of connection related information in the regular message entries, as seen here for instance.
9. Having `LOG_CONNECTION=3` causes the MTA to write this entry. After any messages are dequeued, (the bobby and carl messages in this example), the connection is closed, as indicated by the `C` in this entry.
10. The connection mailhub.sesta.com is closed now that the delivery of the message (dave in this example) is complete.

MTA Logging Example: Inbound Connection Logging

This example illustrates log output for an incoming SMTP message when connection logging is enabled, via `LOG_CONNECTION=3`.

Example MTA Logging: Inbound Connection Logging

```
28-Feb-2007 11:50:59.10 tcp_local + O (1)
TCP|206.184.139.12|25|192.160.253.66|1244 SMTP (2)

28-Feb-2007 11:51:15.12 tcp_local ims-ms EE 1
service@siroe.com rfc822;adam@sesta.com adam@ims-ms-daemon
THOR.SIROE.COM (THOR.SIROE.COM [192.160.253.66]) (3)

28-Feb-2007 11:51:15.32 ims-ms D 1
service@siroe.com rfc822;adam@sesta.com adam@ims-ms-daemon

28-Feb-2007 11:51:15.66 tcp_local + C (4)
TCP|206.184.139.12|25|192.160.253.66|1244 SMTP
```

1. The remote system opens a connection. The `O` character indicates that this entry regards the opening of a connection; the + character indicates that this entry regards an incoming connection.

2. The IP numbers and ports for the connection are shown. In this entry, the receiving system (the system making the log file entry) has IP address 206.184.139.12 and the connection is being made to port 25; the sending system has IP address 192.160.253.66 and is sending from port 1244.
3. In the entry for the enqueue of the message from the incoming TCP/IP channel (`tcp_local`) to the `ims-ms` channel recipient, note that information beyond the default is included since `LOG_CONNECTION=3` is enabled. Specifically, the name that the sending system claimed on its HELO or EHLO line, the sending system's name as found by a DNS reverse lookup on the connection IP number, and the sending system's IP address are all logged; see Configuring Channel Definitions.
4. The inbound connection is closed. The `C` character indicates that this entry regards the closing of a connection; the + character indicates that this entry regards an incoming connection.

Enabling Dispatcher Debugging

Dispatcher error and debugging output (if enabled) are written to the file `dispatcher.log` in the MTA log directory. The dispatcher configuration information is specified in the *msg-svr-base*`/config/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.

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 hand-offs.
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.

To Enable Dispatcher Error Debugging Output

1. Edit the `dispatcher.cnf` file.
2. Set the `DEBUG` option to -1.
   You can also set the logical or environmental variable `IMTA_DISPATCHER_DEBUG` (UNIX), which defines a 32-bit debug mask in hexadecimal, to the value `FFFFFFFF`. The table above describes the meaning of each bit.

To Set Dispatcher Parameters (Oracle Solaris)

The dispatcher services offered in the dispatcher configuration file affects requirements for various system parameters. The system's heap size (`datasize`) must be enough to accommodate the dispatcher's thread stack usage.

1. To display the heap size (that is, default `datasize`), use one of the following:
   The `csh` command:

```
# limit
```

The `ksh` command:

```
# ulimit -a
```

The Solaris utility:

```
# sysdef
```

2. 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.

Managing Message Store, Admin, and Default Service Logs

This section describes logging for the Message Store (POP, IMAP, and HTTP), Admin, and Default services. (See the Messaging Server Log Files table.)

For these services, you specify log settings and to view logs. The settings you specify affect which and how many events are logged. You can use those settings and other characteristics to refine searches for logged events when you are analyzing log files.

This section contains the following subsections:

configutil Logging Parameters

To control the location of log files, use the following `configutil` parameters for specifying directory paths:

> **Note**
> The location of MTA log files, which are in the *msg-svr-base*`/data/log` directory, cannot be modified, but you can change the `log` subdirectory to symbolically link to another location. To separate the MTA logs from the rest of the log files, use `configutil` parameters to specify non-default locations for non-MTA log files.

`configutil` Directory Paths for Log Files

Parameter	Description

`logfile.*.logdir`
`logfile.default.logdir`
`logfile.ens.logdir`
`logfile.http.logdir`
`logfile.imap.logdir`
`logfile.imapproxy.logdir`
`logfile.imta.logdir`
`logfile.metermaid.logdir`
`logfile.mmp.logdir`
`logfile.msgtrace.logdir`
`logfile.pop.logdir`
`logfile.popproxy.logdir`
`logfile.snmp.logdir`
`logfile.submitproxy.logdir`
`logfile.tcp_lmtp_server.logdir`
	Directory path for log files. If this is not specified, log files will be placed in the `msg-install-path/data/log` directory. For the MTA, this option is only used by Message Store insertion tasks Directory path to the imta log file used for Message Store insertion (ims_master, LMTP). It is not used by other parts of the MTA which always log to the default location. The default location is `msg-install-path/data/log`. Changing that path to a soft-link is supported. (Restart of all services required). Syntax: dirpath

Understanding Service Log Characteristics

This section describes the following log characteristics for the message store and administration services: logging levels, categories of logged events, filename conventions for logs, and log-file directories.

Logging Levels

The level, or priority, of logging defines how detailed, or verbose, the logging activity is to be. A higher priority level means less detail; it means that only events of high priority (high severity) are logged. A lower level means greater detail; it means that more events are recorded in the log file.

You can set the logging level separately for each service---POP, IMAP, HTTP, Admin, and Default by setting the `logfile.`*service*`.loglevel` configuration parameter (see Defining and Setting Service Logging Options). You can also use logging levels to filter searches for log events. Table describes the available levels. These logging levels are a subset of those defined by the UNIX `syslog` facility.

Logging Levels for Store and Administration Services

Level	Description
Critical	The minimum logging detail. An event is written to the log whenever a severe problem or critical condition occurs---such as when the server cannot access a mailbox or a library needed for it to run.
Error	An event is written to the log whenever an error condition occurs---such as when a connection attempt to a client or another server fails.
Warning	An event is written to the log whenever a warning condition occurs---such as when the server cannot understand a communication sent to it by a client.
Notice	An event is written to the log whenever a notice (a normal but significant condition) occurs---such as when a user login fails or when a session closes. This is the default log level.
Information	An event is written to the log with every significant action that takes place---such as when a user successfully logs on or off or creates or renames a mailbox.
Debug	The most verbose logging. Useful only for debugging purposes. Events are written to the log at individual steps within each process or task, to pinpoint problems.

When you select a particular logging level, events corresponding to that level and to all higher (less verbose) levels are logged. The default level of logging is `Notice`.

> **Note**
> The more verbose the logging you specify, the more disk space your log files will occupy; for guidelines, see Defining and Setting Service Logging Options.

Categories of Logged Events

Within each supported service or protocol, Messaging Server further categorizes logged events by the facility, or functional area, in which they occur. Every logged event contains the name of the facility that generated it. These categories aid in filtering events during searches. The following table lists the categories that Messaging Server recognizes for logging purposes.

Categories in Which Log Events Occur

Facility	Description
General	Undifferentiated actions related to this protocol or service
LDAP	Actions related to Messaging Server accessing the LDAP directory database
Network	Actions related to network connections (socket errors fall into this category)
Account	Actions related to user accounts (user logins fall into this category)
Protocol	Protocol-level actions related to protocol-specific commands (errors returned by POP, IMAP, or HTTP functions fall into this category)
Stats	Actions related to the gathering of server statistics
Store	Low-level actions related to accessing the message store (read/write errors fall into this category)

For examples of using categories as filters in log searches, see Searching and Viewing Service Logs.

Service Log File Directories

Every logged service is assigned a single directory, in which its log files are stored. All IMAP log files are stored together, as are all POP log files, and log files of any other service. You define the location of each directory, and you also define how many log files of what maximum size are permitted to exist in the directory.

Make sure that your storage capacity is sufficient for all your log files. Log data can be voluminous, especially at lower (more verbose) logging levels.

It is important also to define your logging level, log rotation, log expiration, and server-backup policies appropriately so that all of your log-file directories are backed up and none of them become overloaded; otherwise, you may lose information. See Defining and Setting Service Logging Options.

Understanding Service Log File Format

All message store and administration service log files created by Messaging Server have identical content formats. Log files are multiline text files, in which each line describes one logged event. All event descriptions, for each of the supported services, have the general format:

```
<dateTime> <hostName> <processName>[<pid>]: <category> <logLevel>:
<eventMessage>
```

The following table lists the log file components. Note that this format of event descriptions is identical to that defined by the UNIX `syslog` facility, except that the date/time format is different and the format includes two additional components (*category* and *logLevel*).

Store and Administration Log File Components

POP and IMAP Log File Formats

Component	Definition
dateTime	The date and time at which the event was logged, expressed in *dd/mm/yyyy hh:mm:ss* format, with a time-zone field expressed as *+/-hhmm* from GMT. For example: `02/Jan/1999:13:08:21 -0700`
hostName	The name of the host machine on which the server is running: for example, `showshoe` .*Note:* If there is more than one instance of Messaging Server on the host, you can use the process ID (pid) to separate logged events of one instance from another.
processName	The name of the process that generated the event: for example, `cgi_store`.
pid	The process ID of the process that generated the event: for example, `18753`.
category	The category that the event belongs to: for example, `General` (see Example).
logLevel	The level of logging that the event represents: for example, `Notice` (see Example).
eventMessage	An event-specific explanatory message that may be of any length: for example, `Log created (894305624)`.

Here are three examples of logged events:

```
02/May/1998:17:37:32 -0700 showshoe cgi_store[18753]: General Notice:
Log created (894155852)

04/May/1998:11:07:44 -0400 xyzmail cgi_service[343]: General Error:
function=getserverhello|port=2500|error=failed to connect

03/Dec/1998:06:54:32 +0200 SiroePost imapd[232]: Account Notice: close
[127.0.0.1]
[unauthenticated] 1998/12/3 6:54:32 0:00:00 0 115 0
```

IMAP and POP event entries may end with three numbers. The example above has 0 115 0. The first number is bytes sent by client, the second number is the bytes sent by the server, and third number is mailboxes selected (always 1 for POP).

When viewing a log file in the Log Viewer window, you can limit the events displayed by searching for any specific component in an event, such as a specific logging level or category, or a specific process ID. For more information, see Searching and Viewing Service Logs.

The event message of each log entry is in a format specific to the type of event being logged, that is, each service defines what content appears in any of its event messages. Many event messages are simple and self-evident; others are more complex.

Defining and Setting Service Logging Options

You can define the message store and administration service logging configurations that best serve your administration needs. This section discusses issues that may help you decide on the best configurations and policies, and it explains how to implement them.

Flexible Logging Architecture

The naming scheme for log files (*service.sequenceNum.timeStamp*) helps you to design a flexible log-rotation and backup policy. The fact that events for different services are written to different files makes it easier for you to isolate problems quickly. Also, because the sequence number in a filename is ever-increasing and the timestamp is always unique, later log files do not simply overwrite earlier ones after a limited set of sequence numbers is exhausted. Instead, older log files are overwritten or deleted only when the more flexible limits of age, number of files, or total storage are reached.

Messaging Server supports automatic rotation of log files, which simplifies administration and facilitates backups. You are not required to manually retire the current log file and create a new one to hold subsequent logged events. You can back up all but the current log file in a directory at any time, without stopping the server or manually notifying the server to start a new log file.

In setting up your logging policies, you can set options (for each service) that control limits on total log storage, maximum number of log files, individual file size, maximum file age, and rate of log-file rotation.

Planning the Options You Want

Keep in mind that you must set several limits, more than one of which might cause the rotation or deletion of a log file. Whichever limit is reached first is the controlling one. For example, if your maximum log-file size is 3.5 MB, and you specify that a new log be created every day, you may actually get log files created faster than one per day if log data builds up faster than 3.5 MB every 24 hours. Then, if your maximum number of log files is 10 and your maximum age is 8 days, you may never reach the age limit on log files because the faster log rotation may mean that 10 files will have been created in less than 8 days.

The following default values, provided for Messaging Server administration logs, may be a reasonable starting point for planning:

Maximum number of log files in a directory: 10

Maximum log-file size: 2 MB

Total maximum size permitted for all log files: 20 MB

Minimum free disk space permitted: 5 MB

Log rollover time: 1 day

Maximum age before expiration: 7 days

Level of logging: Notice

You can see that this configuration assumes that server-administration log data is predicted to accumulate at about 2 MB per day, backups are weekly, and the total space allotted for storage of admin logs is at least 25 MB. (These settings may be insufficient if the logging level is more verbose.)

For POP, IMAP or HTTP logs, the same values might be a reasonable start. If all services have approximately the same log-storage requirements as the defaults shown here, you might expect to initially plan for about 150 MB of total log-storage capacity. (Note that this is meant only as a general indication of storage requirements; your actual requirements may be significantly different.)

Understanding Logging Options

You can set options that control the message store logging configuration by the command line.

The optimal settings for these options depend on the rate at which log data accumulates. It may take between 4,000 and 10,000 log entries to occupy 1 MB of storage. At the more verbose levels of logging (such as Notice), a moderately busy server may generate hundreds of megabytes of log data per week. Here is one approach you can follow:

- Set a level of logging that is consistent with your storage limits---that is, a level that you estimate will cause log-data accumulation at approximately the rate you used to estimate the storage limit.
- Define the log file size so that searching performance is not impacted. Also, coordinate it with your rotation schedule and your total storage limit. Given the rate at which log entries accumulate, you might set a maximum that is slightly larger than what you expect to accumulate by the time a rotation automatically occurs. And your maximum file size times your maximum number of files might be roughly equivalent to your total storage limit.
  For example, if your IMAP log rotation is daily, your expected accumulation of IMAP log data is 3 MB per day, and your total storage limit for IMAP logs is 25 MB, you might set a maximum IMAP log-file size of 3.5 MB. (In this example, you could still lose some log data if it accumulated so rapidly that all log files hit maximum size and the maximum number of log files were reached.)
- If server backups are weekly and you rotate IMAP log files daily, you might specify a maximum number of IMAP log files of about 10 (to account for faster rotation if the individual log-size limit is exceeded), and a maximum age of 7 or 8 days.
- Pick a total storage limit that is within your hardware capacity and that coordinates with the backup schedule you have planned for the server. Estimate the rate at which you anticipate that log data will accumulate, add a factor of safety, and define your total storage limit so that it is not exceeded over the period between server backups.
  For example, if you expect to accumulate an average of 3 MB of IMAP log-file data per day, and server backups are weekly, you might specify on the order of 25 - 30 MB as the storage limit for IMAP logs (assuming that your disk storage capacity is sufficient).
- For safety, pick a minimum amount of free disk space that you will permit on the volume that holds the log files. That way, if factors other than log-file size cause the volume to fill up, old log files will be deleted before a failure occurs from attempting to write log data to a full disk.

Searching and Viewing Service Logs

The log files provide the basic interface for viewing message store and administration log data. For a given service, log files are listed in chronological order. Once you have chosen a log file to search, you can narrow the search for individual events by specifying search parameters.

Search Parameters

These are useful search parameters you can specify for viewing log data:

- *A time period*. You can specify the beginning and end of a specific time period to retrieve events from, or you can specify a number of days (before the present) to search. You might typically specify a range to look at logged events leading up to a server crash or other occurrence whose time you know of. Alternatively, you might specify a day range to look at only today's events in the current log file.
- *A level of logging*. You can specify the logging level (see Logging Levels example, Critical to see why the server went down, or Error to locate failed protocol calls.
- *A facility.* You can specify the facility (see Categories of Logged Events that contains the problem; for example, Store if you believe a server crash involved a disk error, or Protocol if the problem lies in an IMAP protocol command error.
- *A text search pattern.* You can provide a text search pattern to further narrow the search. You can include any component of the event (see Understanding Service Log File Format search, such as event time, process name, process ID, and any part of the event message (such as remote host name, function name, error number, and so on) that you know defines the event or events you

want to retrieve.
Your search pattern can include the following special and wildcard characters:

* Any set of characters (example: `*.com`)

? Any single character (example: `199?`)

[*nnn*] Any character in the set *nnn* (example: `[aeiou]`)

[] Any character not in the set *nnn* (example: `[aeiou]`)

[] Any character in the range *n-m* (example: `[A-Z]`)

[*n-m*] Any character not in the range *n-m* (example: `[0-9]`)

}} Escape character: place before {{*, ?, [, or] to use them as literals

Note: Searches are case-sensitive.

Examples of combining logging level and facility in viewing logs might include the following:

- Specifying Account facility (and Notice level) to display failed logins, which may be useful when investigating potential security breaches
- Specifying Network facility (and all logging levels) to investigate connection problems
- Specifying all facilities (and Critical logging level) to look for basic problems in the functioning of the server

Working With Service Logs

This section describes how to work with service logs by using the `configutil` command for searching and viewing logs.

To Send Service Logs to syslog

1. Run the `configutil` command with the `syslogfacility` option:
   `configutil -o logfile.`*service*`.syslogfacility -v` *value*
   where *service* is `admin`, `pop`, `imap`, `imta`, or `http` and *value* is `user`, `mail`, `daemon`, `local0` to `local7`, or none.
   Once the value is set, messages are logged to the `syslog` facility corresponding to the set value and all the other log file service options are ignored. When the option is not set or the value is `none`, logging uses the Messaging Server log files.

To Disable HTTP Logging

If your system does not support HTTP message access, that is, Webmail, you can disable HTTP logging by setting the following variables. Do not set these variables if your system requires Webmail support (for example, Messenger Express).

1. Run the following `configutil` commands:

```
configutil -o service.http.enable -v no
configutil -o service.http.enablesslport -v no
```

To Set the Server Log Level

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.loglevel -v <level>
```

   where *service* is `admin`, `pop`, `imap`, `imta`, or `http` and *loglevel* is `Nolog`, `Critical`, `Error`, `Warning`, `Notice`, `Information`, or `Debug`.

To Specify a Directory Path for Server Log Files

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.logdir -v <dirpath>
```

To Specify a Maximum File Size for Each Service Log

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.maxlogfilesize -v <size>
```

   where *size* specifies a number of bytes.

To Specify a Service Log Rotation Schedule

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.rollovertime -v <number>
```

   where *number* specifies a number of seconds.

To Specify a Maximum Number of Service Log Files Per Directory

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.maxlogfiles -v <number>
```

   where *number* specifies the maximum number of log files.

To Specify a Storage Limit

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.maxlogsize -v <number>
```

   where *number* specifies a number in bytes.

To Specify the Minimum Amount of Free Disk Space to Reserve

1. Run the following `configutil` command:

```
configutil -o logfile.<service>.minfreediskspace -v <number>
```

   where *number* specifies a number in bytes.

To Specify an Age for Logs at Which They Expire

```
configutil -o logfile.<service>.expirytime -v <number>
```

where *number* specifies a number in seconds.

Implementing and Configuring Message Store Transaction Logging

> **Note**
> You can implement and configure Message Store Store transaction logging only if you are running a unified configuration. See *Messaging Server Unified Configuration System Administrator's Guide* for more information.

Using Message Tracing for Message Store Logging

You can use Message Store logging to trace messages by message ID in a way similar to how the MTA traces messages. Tracing messages in this fashion enables you to track the critical events of a message's life cycle.

To trace messages in the Message Store logs, you need to configure message tracing in addition to the normal logging configuration. By default, message tracing is not enabled.

> **Note**
> Message tracing will fill up a large amount of disk space. Do not enable this feature unless you have adequate disk space.

Message Store logging can track the following operations:

- IMAP/POP login events.
  - Client IP address and port
  - login identity
  - authentication type
  - session id (IMAP only)
- `append` - `append` is the primary way the Message Store library adds a message to a folder. Tracing `append` shows messages entering the Message Store. An `append` logs the following information:
  - userid
  - mailbox name
  - message size
  - Message-id

- `fetch` - The IMAP command that retrieves a message or part of a message for the end user. Fetch commands that access only the message header are not traced. Only fetch commands that access the message body are traced. A `fetch` logs the following information:
  - imap userid
  - mailbox name
  - offset
  - size
  - Message-id

- `expunge` - An IMAP term that is extended in this case to reference when any service removes a message from a user's folder. An `expunge` logs the following information:
  - mailbox name
  - Message-id

To Enable Message Tracing

1. Run either of the following `configutil` commands:

```
configutil -o local.msgtrace.active -v yes
configutil -o logfile.imap.loglevel -v info
```

   Message trace information is written to the default log for each process. The IMAP fetches appear in the `imap` log file. The `ims_master` appends appear in the `ims_master` channel log file.

```
configutil -o local.msgtrace.active -v msgtrace
configutil -o logfile.msgtrace.loglevel -v info
```

   In this command message trace information for all processes is written to the `msgtrace` log file

2. restart imapd

```
stop-msg imap
start-msg imap
```

To Configure LMTP Logging

1. If you are using LMTP, and not using a single "msgtrace" log file, then you must also locally configure the `tcp_lmtp_server` log file. If you are not using LMTP, or are not using message trace, or are using message trace in the "msgtrace" log file, you do not need to initialize the LMTP Message Store side log. (LMTP already logs MTA information separately.) For example:

```
configutil -o "local.logfile.tcp_lmtp_server.buffersize" -v "0"
configutil -o "local.logfile.tcp_lmtp_server.expirytime" -v "604800"
configutil -o "local.logfile.tcp_lmtp_server.flushinterval" -v "60"
configutil -o "local.logfile.tcp_lmtp_server.logdir" -v \
"/opt/SUNWmsgsr/data/log"
configutil -o "local.logfile.tcp_lmtp_server.loglevel" -v "Information"
configutil -o "local.logfile.tcp_lmtp_server.logtype" -v "NscpLog"
configutil -o "local.logfile.tcp_lmtp_server.maxlogfiles" -v "10"
configutil -o "local.logfile.tcp_lmtp_server.maxlogfilesize" -v
"2097152"
configutil -o "local.logfile.tcp_lmtp_server.maxlogsize" -v "20971520"
configutil -o "local.logfile.tcp_lmtp_server.minfreediskspace" \
-v "5242880"
configutil -o "local.logfile.tcp_lmtp_server.rollovertime" -v "86400"
```

Other Message Store Logging Features

Messaging Server provides a feature called telemetry that can capture a user's entire IMAP or POP session into a file. This feature is useful for debugging client problems. For example, if a user complains that their message access client is not working as expected, this feature can be used to trace the interaction between the access client and Messaging Server. See in Checking User IMAP/POP/Webmail Session by Using Telemetry.

Message Store Logging Examples

The exact field format and list of fields logged in the Message Store log files vary according to the logging options set. This section shows a few examples of interpreting typical sorts of log entries.

Message Store Logging Example: Bad Password

When a user types an invalid password, "authentication" failure is logged, as opposed to a "user not found" message. The message "user not found" is the text passed to the client for security reasons, but the real reason (invalid password) is logged.

Example Message Store Logging: Invalid Password

```
[30/Aug/2004:16:53:05 -0700] vadar imapd[13027]: Account Notice:
badlogin:
[192.18.126.64:40718] plaintext user1 authentication failure
```

Message Store Logging Example: Account Disabled

The following example shows why a user cannot log in due to a disabled account. Furthermore, the disabled account is clarified as "(inactive)" or "(hold)."

Example Message Store Logging: Account Disabled

```
[30/Aug/2004:16:53:31 -0700] vadar imapd[13027]: Account Notice:
badlogin:
[192.18.126.64:40720] plaintext user3 account disabled (hold)
```

Message Store Logging Example: Message Appended

The following example shows an append message, which occurs when whenever a message is appended to a folder. The Message Store log records all messages entering the Message Store through the `ims_master` and `lmtp` channels. Records the "append" of user ID, folder, message size, and message ID.

Example Message Store Logging: Append

```
[31/Aug/2004:16:33:14 -0700] vadar ims_master[13822]: Store
Information:append:
user1:user/user1:659:&lt;Roam.SIMC.2.0.6.1093995286.11265.user1@vadar.siroe
```

Message Store Logging Example: Message Retrieved by a Client

The Message Store log writes a "fetch" message when a client retrieves a message. The Message Store log records all client fetches of at least one body part. Records the "fetch" of user ID, folder, and message-ID.

Example Message Store Logging: Message Retrieved by a Client

```
[31/Aug/2004:15:55:26 -0700] vadar imapd[13729]: Store Information:
fetch:user1:user/user1:&lt;Roam.SIMC.2.0.6.1093051161.3655.user1@vad.siroe.
```

Message Store Logging Example: Message Removed from a Folder

Example Message Store Logging Example: Message Removed from a Folder

The Message Store writes an "expunge" message when an IMAP or POP message is removed from a folder (but not removed from the system). It is logged whether it is expunged by the user or a utility. Records the "expunge" of folder and message ID.

```
31/Aug/2004:16:57:36 -0700] vadar imexpire[13923]: Store Information:
expunge:user/user1:<Roam.SIMC.2.0.6.1090458838.2929.user1@vadar.siroe.com>
```

Message Store Logging Example: Duplicate Login Messages

If you configure message trace for one msgtrace log file, the normal "login" messages, which appear in the imap and pop log files, are duplicated in the msgtrace file. The following is a normal login message:

Example Message Store Logging: Login

```
[30/Aug/2004:16:53:13 -0700] vadar imapd[13027]: Account Information: login
[192.18.126.64:40718] user1 plaintext
```

Using Message Store Log Messages

Use the following links to significant Message Store logging and error message pages:

- **Message Store Log Messages and Appropriate Actions.** Lists significant log messages and provides guidance in interpreting and addressing them.
- **Overview of Messaging Server Logging.** Provides an introduction to Messaging Server logging concepts.
- **Tools for Managing Logs.** Lists available tools for managing logs.
- **`configutil` Logging Parameters.** Search for `logfile`.
- **General Discussion on Message Store Logging.** Describes logging for the Message Store (POP, IMAP, and HTTP), Admin, and Default services.

Message Store Log Message Categories

Logging messages with similar issues and attributes are grouped into categories. Category pages provide additional information about how to deal with the log messages. Each log message provides a link to its category page.

A link to each of the category pages is listed below. If you have useful knowledge or experience with specific log messages, we encourage you to add any tips or notes to the appropriate pages.

Message Store Logging - archive
Message Store Logging - config
Message Store Logging - cert
Message Store Logging - config
Message Store Logging - counters
Message Store Logging - database
Message Store Logging - dblock
Message Store Logging - ens
Message Store Logging - imapd
Message Store Logging - imaplib
Message Store Logging - ims_master
Message Store Logging - jmq
Message Store Logging - ldap
Message Store Logging - libstore
Message Store Logging - mailwave
Message Store Logging - msghashdb
Message Store Logging - mshttpd
Message Store Logging - mqueue
Message Store Logging - msprobe
Message Store Logging - pab
Message Store Logging - plugin
Message Store Logging - popd
Message Store Logging - recovery
Message Store Logging - sasl
Message Store Logging - sessiondb
Message Store Logging - smime
Message Store Logging - snapshots
Message Store Logging - ssl
Message Store Logging - sso
Message Store Logging - stored
Message Store Logging - utils

MMP Logging

See MMP Reference.

Chapter 30. Troubleshooting the MTA

Troubleshooting the MTA

This information describes common tools, methods, and procedures for troubleshooting the Message Transfer Agent (MTA).

Topics:

A related topic, monitoring procedures can be found in Monitoring Messaging Server.

> **Note**
> This information assumes that you are familiar with the MTA, both from a conceptual and administration perspective.

Troubleshooting Overview

One of the first steps in troubleshooting the MTA is to determine where to begin the diagnosis. Depending on the problem, you might look for error messages in log files. In other situations, you might check all of the standard MTA processes, review the MTA configuration, or start and stop individual channels. Whatever approach you use, consider the following questions when troubleshooting the MTA:

- Did configuration or environmental problems prevent messages from being accepted (for example, disk space or quota problems)?
- Were MTA services such as the Dispatcher and the Job Controller present at the time the message entered the message queue?
- Did network connectivity or routing problems cause messages to be stuck or misrouted on a remote system?
- Did the problem occur before or after a message entered into the message queue?

This information addresses these questions in the subsequent sections.

Standard MTA Troubleshooting Procedures

This section outlines standard troubleshooting procedures for the MTA. Follow these procedures if a problem does not generate an error message, if an error message does not provide enough diagnostic information, or if you want to perform general wellness checks, testing, and standard maintenance of the MTA.

- An MTA Troubleshooting Example

Check the MTA Configuration

Test your address configuration by using the `imsimta test -rewrite` utility. With this utility, you can test the MTA's address rewriting and channel mapping without actually having to send a message. Refer to the section on message transfer agent command-line utilities in *Messaging Server Administration Reference* for more information.

The utility will normally show address rewriting that will be applied as well as the channel to which messages will be queued. However, syntax errors in the MTA configuration will cause the utility to issue an error message. If the output is not what you expect, you may need to correct your configuration.

Check the Message Queue Directories

Check if messages are present in the MTA message queue directory, typically *msg-svr-base* `/data/queue/`. Use command-line utilities like `imsimta qm` to check for the presence of expected message files under the MTA message queue directory. For more information, see `imsimta qm` and `imsimta qm counters`.

If the `imsimta test -rewrite` output looks correct, check that messages are actually being placed in the MTA message queue subdirectories. To do so, enable message logging and check the log files in the directory *msg-svr-base*/log/. For more information on MTA logging, see MTA Messaging and Connection Logs. You can track a specific message by its message ID to ensure that it is being placed in the MTA message queue subdirectories. If you are unable to find the message, you may have a problem with file disk space or directory permissions.

Check the Ownership of Critical Files

You should have selected a mail server user account (`mailsrv` by default) when you installed Messaging Server. The following directories, subdirectories, and files should be owned by this account:

```
<msg-svr-base>/data/queue/
<msg-svr-base>/data/log
<msg-svr-base>/data/tmp
```

Commands, like the ones in the following UNIX system example, can be used to check the protection and ownership of these directories:

> The following applies starting with Messaging Server 7.

```
ls -l -p -d /opt/sun/comms/messaging64/data/queue
drwx------   2 mailsrv  mail 512 Sep  18 21:17
/opt/sun/comms/messaging64/data/queue

ls -l -p -d /opt/sun/comms/messaging64/data/log
drwx------   2 mailsrv  mail  2560 Oct 15 05:25
/opt/sun/comms/messaging64/data/log

ls -l -p -d /opt/sun/comms/messaging64/data/tmp
drwx------   2 mailsrv  mail   512 Sep  18 21:17
/opt/sun/comms/messaging64/data/tmp
```

The following output applies to Messaging Server 6.3.

```
ls -l -p -d /opt/SUNWmsgsr/data/queue
drwxr-x---   2 mailsrv  mail 512 Jan  4 16:09 /opt/SUNWmsgsr/data/queue/

ls -l -p -d /opt/SUNWmsgsr/data/log
drwxr-x---   2 mailsrv  mail  3072 Feb 16 12:07 /opt/SUNWmsgsr/data/log/

ls -l -p -d /opt/SUNWmsgsr/data/tmp
drwxr-x---   2 mailsrv  mail   512 Feb 16 12:55 /opt/SUNWmsgsr/data/tmp/
```

Check that the files in *msg-svr-base*`/data/queue` are owned by the MTA account by a using command such as the following UNIX system example:

The following applies starting with Messaging Server 7.

```
ls -l -p -R /opt/sun/comms/messaging64/data/queue
```

Check that the Job Controller and Dispatcher Are Running

The MTA Job Controller handles the execution of the MTA processing jobs, including most outgoing (master) channel jobs.

Some MTA channels, such as the MTA's multi-threaded SMTP channels, include resident server processes that process incoming messages. These servers handle the slave (incoming) direction for the channel. The MTA Dispatcher handles the creation of such MTA servers. Dispatcher configuration options control the availability of the servers, the number of created servers, and how many connections each server can handle.

To check that the Job Controller and Dispatcher are present, and to see if there are MTA servers and processing jobs running, use the command `imsimta process`. Under idle conditions the command should result in `job_controller` and `dispatcher` processes. For example:

The following output applies to Messaging Server 6.3.

```
# imsimta process
USER      PID S VSZ    RSS   STIME    TIME     COMMAND
mailsrv 9567 S 18416 9368  02:00:02  0:00  /opt/SUNWmsgsr/lib/tcp_smtp_server
mailsrv 6573 S 18112 5720  Jul_13    0:00  /opt/SUNWmsgsr/lib/job_controller
mailsrv 9568 S 18416 9432  02:00:02  0:00  /opt/SUNWmsgsr/lib/tcp_smtp_server
mailsrv 6574 S 17848 5328  Jul_13    0:00  /opt/SUNWmsgsr/lib/dispatcher
```

If the Job Controller is not present, the files in the *msg-svr-base*`/data/queue` directory will get backed up and messages will not be delivered. If you do not have a Dispatcher, then you will be unable to receive any SMTP connections.

See `imsimta process` for more information.

You could also use `imsimta qm jobs` to list, channel by channel, all active and pending delivery processing jobs currently being managed by the Job Controller. Additional cumulative information is provided for each channel such as the number of message files successfully delivered and those requeued for subsequent delivery attempts. The command syntax is as follows:

```
jobs [-[no]hosts] [-[no]jobs] [-[no]messages] [channel-name]
```

If neither the Job Controller nor the Dispatcher is present, you should review the `dispatcher.log-*` or `job_controller.log-*` file in the *msg-svr-base*`/data/log` directory.

If the log files do not exist or do not indicate an error, start the processes by using the `start-msg` command.

> **Note**
> You should not see multiple instances of the Dispatcher or Job Controller when you run `imsimta process`, unless the system is in the process of forking (`fork()`) child processes before it executes (`exec()`) the program that needs to run. However, the time frame during such duplication is very small.

Check the Log Files

If MTA processing jobs run properly but messages stay in the message queue directory, you can examine the log files to see what is happening. All MTA log files are created in the *msg-svr-base*`/log` directory. Log file name formats for various MTA processing jobs are shown in the following table.

MTA Log Files

File Name	Log File Contents
channel`master.log-`*_uniqueid*	Output of master program (usually client) for *channel.*
channel`slave.log-`*_uniqueid*	Output of slave program (usually server) for *channel.*
`dispatcher.log-`*uniqueid*	Dispatcher debugging. This log is created regardless if the Dispatcher `DEBUG` option is set. However, to get detailed debugging information, you should set the `DEBUG` option to a non-zero value.
`imta`	`ims-ms` channel error messages when there is a problem in delivery.
`job_controller.log-`*uniqueid*	Job controller logging. This log is created regardless if the Job Controller `DEBUG` option is set. However, to get detailed debugging information, you should set the `DEBUG` option to a non-zero value.
`tcp_smtp_server.log-`*uniqueid*	Debugging for the `tcp_smtp_server`. The information in this log is specific to the server, not to messages.
`return.log-`*uniqueid*	Debug output for the periodic MTA message bouncer job; this log file is created if the `return_debug` option is used in the `option.dat`.

Note
Each log file is created with a unique ID (*uniqueid*) to avoid overwriting an earlier log created by the same channel. To find a specific log file, you can use the `imsimta view` utility. You can also purge older log files by using the `imsimta purge` command. Note, however, that by default this command is run on a regular basis (see Pre-defined Automatic Tasks). For more information, see imsimta purge.

The *channel*`master.log-`*uniqueid* and *channel*`slave.log-`*uniqueid* log files will be created in any of the following situations:

- There are errors in your current configuration.
- The `master_debug` or `slave_debug` keywords are set on the channel in the `imta.cnf` file.
- If `mm_debug` is set to a non-zero value (`mm_debug > 0`) in your `option.dat` file in *msg-svr-base*`/config/` directory.

For more information on debugging channel master and slave programs, see *Messaging Server Administration Reference*.

Run a Channel Program Manually

When diagnosing an MTA delivery problem it is helpful to manually run an MTA delivery job, particularly after you enable debugging for one or more channels.

The command `imsimta submit` notifies the MTA Job Controller to run the channel. If debugging is enabled for the channel in question, `imsimta submit` creates a log file in the *msg-svr-base*`/log` directory as shown in MTA Log Files.

The command `imsimta run` performs outbound delivery for the channel under the currently active process, with output directed to your terminal. This might be more convenient than submitting a job, particularly if you suspect problems with job submission itself.

Note
To manually run channels, the Job Controller must be running.

For information on syntax, options, parameters, examples of `imsimta submit` and `imsimta run` commands, refer to *Messaging Server Administration Reference*.

Starting and Stopping Individual Channels

In some cases, stopping and starting individual channels may make message queue problems easier to diagnose and debug. Stopping a message queue allows you to examine queued messages to determine the existence of loops or spam attacks.

To Stop Outbound Processing (dequeueing) for a Specific Channel

1. Use the `imsimta qm stop` command to stop a specific channel. Doing so prevents you from having to stop the Job Controller and having to recompile the configuration. In the following example, the `conversion` channel is stopped:
   `imsimta qm stop conversion`
2. To resume processing, use the `imsimta qm start` command to restart the channel. In the following example, the `conversion` channel is started:

`imsimta qm start conversion`
See `imsimta qm` for more information on `imsimta qm start` and `imsimta qm stop` commands.

> **Note**
> The command `imsimta qm start/stop` *channel* might fail if run simultaneously for many channels at the same time. The tool might have trouble updating the `hold_list` and could report: `QM-E-NOTSTOPPED, unable to stop the channel; cannot update the hold list`. `imsimta qm start/stop` *channel* should only be used sequentially with a few seconds interval between each run.
>
> If you only want the channel to run between certain hours, use the following options in the channel definition section in the job controller configuration file:
>
> ```
> urgent_delivery=08:00-20:00
> normal_delivery=08:00-20:00
> nonurgent_delivery=08:00-20:00
> ```

To Stop Inbound Processing from a Specific Domain or IP Address (Enqueuing to a Channel)

You can run one of the following processes if you want to stop inbound message processing for a specific domain or IP address, while returning temporary SMTP errors to client hosts. By doing so, messages are not held on your system. Refer to the PART 1. MAPPING TABLES.

- To stop inbound processing for a specific host or domain name, add the following access rule to the ORIG_SEND_ACCESS mapping table in the MTA mappings file (typically *msg-svr-base* `/config/mappings`):

```
ORIG_SEND_ACCESS

  *|*@sesta.com|*|*        $X4.2.1|$NHost$ temporarily$ blocked
```

By using this process, the sender's remote MTA will hold messages on their systems, continuing to resend them periodically until you restart inbound processing.

- To stop inbound processing for a specific IP address, add the following access rule to the PORT_ACCESS mapping table in the MTA mappings file (typically *msg-svr-base* `/config/mappings`):

```
PORT_ACCESS

    TCP|*|25|_IP_address_to_block_|*    $N500$ can't$ connect$ now
```

When you want to restart inbound processing from the domain or IP address, be sure to remove these rules from the mapping tables and recompile your configuration. In addition, you may want to create unique error messages for each mapping table. Doing so will enable you to determine which mapping table is being used.

An MTA Troubleshooting Example

This section explains how to troubleshoot a particular MTA problem step-by-step. In this example, a mail recipient did not receive an attachment to an email message. Note: In keeping with MIME protocol terminology, the "attachment" is referred to as a "message part" in this section. The aforementioned troubleshooting techniques are used to identify where and why the message part disappeared (See Standard MTA Troubleshooting Procedures). By using the following steps, you can determine the path the message took through the MTA. In addition, you can determine if the message part disappeared before or after the message entered the message queue. To do so, you will need to manually stop and run channels, capturing the relevant files.

> **Note**
> The Job Controller must be running when you manually run messages through the channels.

Identify the Channels in the Message Path

By identifying which channels are in the message path, you can apply the `master_debug` and `slave_debug` keywords to the appropriate channels. These keywords generate debugging output in the channels' master and slave log files. In turn, the master and slave debugging information will assist in identifying the point where the message part disappeared.

1. Add `log_message_id=1` in the `option.dat` file in directory *msg-svr-base*`/config`. With this parameter, you will see message ID: header lines in the `mail.log_current` file.
2. Run `imsimta cnbuild` to recompile the configuration{{.}}
3. Run `imsimta restart dispatcher` to restart the SMTP server.
4. Have the end user resend the message with the message part.
5. Determine the channels that the message passes through.
   While there are different approaches to identifying the channels, the following approach is recommended:
   a. On UNIX platforms, use the `grep` command to search for message ID: header lines in the `mail.log_current` file in directory *msg-svr-base*`/log`.
   b. Once you find the message ID: header lines, look for the E (enqueue) and D (dequeue) records to determine the path of the message. Refer to Understanding the MTA Log Entry Format for more information on logging entry codes. See the following E and D records for this example:

```
29-Aug-2001 10:39:46.44  _tcp_local_ _conversion_        E 2
...
29-Aug-2001 10:39:46.44  _conversion_ _tcp_intranet_     E 2
...
29-Aug-2001 10:39:46.44  _tcp_intranet_                 D 2
...
```

The channel on the left is the source channel, and the channel on the right is the destination channel. In this example, the E and D records indicate that the message's path went from the `tcp_local` channel to the `conversion` channel and finally to the `tcp_intranet` channel.

Manually Start and Stop Channels to Gather Data

This section describes how to manually start and stop channels. See Starting and Stopping Individual Channels By starting and stopping the channels in the message's path, you are able to save the message and log files at different stages in the MTA process. These files are later used to To Identify the Point of Message Breakdown.

To Manually Start and Stop Channels

1. Set the `mm_debug=5` in the `option.dat` file in directory *msg-svr-base*`/config` to provide substantial debugging information.
2. Add the `slave_debug` and `master_debug` keywords to the appropriate channels in the `imta.cnf` file in the *msg-svr-base*`/config` directory.
    a. Use the `slave_debug` keyword on the inbound channel (or any channel where the message is switched to during the initial dialog) from the remote system that is sending the message with the message part. In this example, the `slave_debug` keyword is added to the `tcp_local` channel.
    b. Add the `master_debug` keyword to the other channels that the message passed through and were identified in Identify the Channels in the Message Path would be added to the `conversion` and `tcp_intranet` channels.
    c. Run the command `imsimta restart dispatcher` to restart the SMTP server.
3. Use the `imsimta qm stop` and `imsimta qm start` commands to manually start and stop specific channels. For more on information by using these keywords, see Starting and Stopping Individual Channels.
4. To start the process of capturing the message files, have the end user resend the message with the message part.
5. When the message enters a channel, the message will stop in the channel if it has been stopped with the `imsimta qm stop` command. For more information, see Step Step 3.
    a. Copy and rename the message file before you manually run the next channel in the message's path. See the following UNIX platform example:
    `# cp ZZ01K7LXW76T7O9TD0TB.00 ZZ01K7LXW76T7O9TD0TB.KEEP1`
    The message file typically resides in directory similar to *msg-svr-base*`/data/queue/`*destination_channel*`/001`. The *destination_channel* is the next channel that the message passes through (such as: `tcp_intranet`). If you want to create subdirectories (like `001`, `002`, and so on) in the *destination_channel* directory, add the `subdirs` keyword to the channels.
    b. It is recommended that you number the extensions of the message each time you trap and copy the message to identify the order in which the message is processed.
6. Resume message processing in the channel and enqueue to the next destination channel in the message's path. To do so, use the `imsimta qm start` command.
7. Copy and save the corresponding channel log file (for example: `tcp_intranet_master.log-*`) located in the *msg-svr-base*`/log` directory. Choose the appropriate log file that has the data for the message you are tracking. Make sure that the file you copy matches the timestamp and the subject header for the message as it comes into the channel. In the example of the `tcp_intranet_master.log-*`, you might save the file as `tcp_intranet_master.keep` so the file is not deleted.
8. Repeat steps 5 - 7 until the message has reached its final destination.
The log files you copied in Step Step 7 should correlate to the message files that you copied in Step Step 5. If, for example, you stopped all of the channels in the missing message part scenario, you would save the `conversion_master.log-*` and the `tcp_intranet_master.log-*` files. You would also save the source channel log file `tcp_local_slave.log-*`. In addition, you would save a copy of the corresponding message file from each destination channel: `ZZ01K7LXW76T7O9TD0TB.KEEP1` from the `conversion` channel and `ZZ01K7LXW76T7O9TD0TB.KEEP2` from the `tcp_intranet` channel{{.}}
9. Remove debugging options once the message and log files have been copied.
    a. Remove the `slave_debug` and the `master_debug` keywords from the appropriate channels in the `imta.cnf` file in the *msg-svr-base*`/config` directory.
    b. Reset the `mm_debug=0,` and remove `log_message_id=1` in the `option.dat` file in the *msg-svr-base*`/config` directory.
    c. Recompile the configuration by using `imsimta cnbuild.`
    d. Run the command `imsimta restart dispatcher` to restart the SMTP server.

To Identify the Point of Message Breakdown

1. By the time you have finished starting and stopping the channel programs, you should have the following files with which you can use to troubleshoot the problem:

a. All copies of the message file (for example: `ZZ01K7LXW76T7O9TD0TB.KEEP1`) from each channel program
b. A `tcp_local_slave.log-*` file
c. A set of *channel* `_master.log-*` files for each destination channel
d. A set of `mail.log_current` records that show the path of the message
All files should have timestamps and message ID values that match the message ID: header lines in the `mail.log_current` records. Note that the exception is when messages are bounced back to the sender; these bounced messages will have a different message ID value than the original message.

2. Examine the `tcp_local_slave.log-*` file to determine if the message had the message part when it entered the message queue.
Look at the SMTP dialog and data to see what was sent from the client machine.
If the message part did not appear in the `tcp_local_slave.log-*` file, then the problem occurred before the message entered the MTA. As a result, the message was enqueued without the message part. If this the case, the problem could have occurred on the sender's remote SMTP server or in the sender's client machine.
3. Investigate the copies of the message files to see where the message part was altered or missing.
If any message file showed that the message part was altered or missing, examine the previous channel's log file. For example, you should look at the `conversion_master.log-*` file if the message part in the message entering the `tcp_intranet` channel was altered or missing.
4. Look at the final destination of the message.
If the message part looks unaltered in the `tcp_local_slave.log`, the message files (for example: `ZZ01K7LXW76T7O9TD0TB.KEEP1`), and the *channel*`_master.log-*` files, then the MTA did not alter the message and the message part is disappearing at the next step in the path to its final destination.
If the final destination is the `ims-ms` channel (the Message Store), then you might download the message from the server to a client machine to determine if the message part is being dropped during or after this transfer. If the destination channel is a `tcp_*` channel, then you need to go to the MTA in the message's path. Assuming it is an Messaging Server MTA, you will need to repeat the entire troubleshooting process (See Identify the Channels in the Message Path, Manually Start and Stop Channels to Gather Data, and this section). If the other MTA is not under your administration, then the user who reported the problem should contact that particular site.

Common MTA Problems and Solutions

This sections lists common problems and solutions for MTA configuration and operation.

- TLS Problems
- Changes to Configuration Files or MTA Databases Do Not Take Effect
- The MTA Sends Outgoing Mail but Does Not Receive Incoming Mail
- Dispatcher (SMTP Server) Won't Start Up
- Timeouts on Incoming SMTP connections
- Messages Are Not Dequeued
- MTA Messages Are Not Delivered
- Messages are Looping
- Received Message is Encoded
- Server-Side Rules (SSR) Are Not Working
- Slow Response After Users Press Send Email Button
- Asterisks in the Local Parts of Addresses or Received Fields
- Abnormal Job Controller Terminations Seen in `job_controller` Logs

TLS Problems

If, during SMTP dialog, the `STARTTLS` command returns the following error:

```
454 4.7.1 TLS library initialization failure
```

and if you have certificates installed and working for POP and IMAP access, check the following:

- Protections/ownerships of the certificates have to be set so `mailsrv` account can access the files.
- The directory where the certificates are stored need to have protections/ownerships set such that the `mailsrv` account can access the files within that directory.

After changing protections and installing certificates, you must run:

```
stop-msg dispatcher
start-msg dispatcher
```

Restarting should work, but it is better to shut it down completely, install the certificates, and then start things back up.

Changes to Configuration Files or MTA Databases Do Not Take Effect

If changes to your configuration, mapping, conversion, security, option, or alias files are not taking effect, check to see if you have performed the following steps:

1. Recompile the configuration (by running `imsimta cnbuild`).
2. Restart the appropriate processes (like `imsimta restart dispatcher`).
3. Re-establish any client connections.

The MTA Sends Outgoing Mail but Does Not Receive Incoming Mail

Most MTA channels depend upon a slave or channel program to receive incoming messages. For some transport protocols that are supported by the MTA (like TCP/IP and UUCP), you need to make sure that the transport protocol activates the MTA slave program rather than its standard server. Replacing the native `sendmail` SMTP server with the MTA SMTP server is performed as a part of the Messaging Server installation.

For the multi-threaded SMTP server, the startup of the SMTP server is controlled by the Dispatcher. If the Dispatcher is configured to use a `MIN_PROCS` value greater than or equal to one for the SMTP service, then there should always be at least one SMTP server process running (and potentially more, according to the `MAX_PROCS` value for the SMTP service). The `imsimta process` command may be used to check for the presence of SMTP server processes. See `imsimta process` in *Messaging Server Administration Reference* for more information.

Dispatcher (SMTP Server) Won't Start Up

If the dispatcher won't start up, first check the `dispatcher.log-*` for relevant error messages. If the log indicates problems creating or accessing the `/tmp/.`*`path`*`.dispatcher.socket` file, then verify that the `/tmp` protections are set to 1777. This would show up in the permissions as follows:

```
drwxrwxrwt 8 root sys 734 Sep 17 12:14 tmp/
.
```

Also do an `ls -l` of the *`path.version-specific-name`*`.dispatcher.socket` file and confirm the proper ownership. For example, if this is created by `root`, then it is inaccessible by `inetmail`.

Do not remove the *`.path`*`.dispatcher.file` and do not create it if it's missing. The dispatcher will create the file. If protections are not set to 1777, the dispatcher will not start or restart because it won't be

able to create/access the socket file. In addition, there may be other problems occurring not related to the Messaging Server.

Messaging Server 6: `/tmp/.SUNWmsgsr.dispatcher.socket`
Messaging Server 7 32-bit: *msg-svr-base* `/config/.var_opt_sun_comms_messaging_config.dispatcher.socket`
Messaging Server 7 64-bit: *msg-svr-base* `/config/.var_opt_sun_comms_messaging64_config.dispatcher.socket`

Timeouts on Incoming SMTP Connections

Timeouts on incoming SMTP connections are most often related to system resources and their allocation. The following techniques can be used to identify the causes of timeouts on incoming SMTP connections.

To Identify the Causes of Timeouts on Incoming SMTP Connections

1. Check how many simultaneous incoming SMTP connections you allow. This is controlled by the `MAX_PROCS` and `MAX_CONNS` Dispatcher settings for the SMTP service. The number of simultaneous connections allowed is `MAX_PROCS*MAX_CONNS`. If you can afford the system resources, consider raising this number if it is too low for your usage.
2. Another technique you can use is to open a TELNET session.
   In the following example, the user connects to `127.0.0.1` port 25. Once connected, 220 banner is returned. For example:

   ```
   telnet 127.0.0.1 25
   Trying 127.0.0.1...
   Connected to 127.0.0.1.
   Escape character is '^]'.
   220 budgie.sesta.com --Server ESMTP (Sun Java System Messaging
   Server 6.1
   (built May  7 2001))
   ```

   If you are connected and receive a 220 banner, but additional commands (like `ehlo` and `mail from`) do not illicit a response, then you should run `imsimta test -rewrite` to ensure that the configuration is correct.

3. If the response time of the 220 banner is slow, and if running the `pstack` command on the SMTP server shows the following `iii_res*` functions (these functions indicate that a name resolution lookup is being performed):

   ```
   febe2c04 iii_res_send (fb7f4564, 28, fb7f4de0, 400, fb7f458c,
   fb7f4564) +
   42c febdfdcc iii_res_query (0, fb7f4564, c, fb7f4de0, 400, 7f) +
   254
   ```

   then it is likely that the host has to do reverse name resolution lookups, even on a common pair like `localhost/127.0.0.1`. To prevent such a performance slowdown, you should reorder your host's lookups in the `/etc/nsswitch.conf` file. To do so, change the following line in the `/etc/nsswitch.conf` file from:

```
hosts: dns nis [NOTFOUND=return] files
```

to:

```
hosts: files dns nis [NOTFOUND=return]
```

Making this change in the `/etc/nsswitch.conf` file can improve performance as fewer SMTP servers have to handle messages instead of multiple SMTP servers having to perform unnecessary lookups.

4. You can also put the `slave_debug` keyword on the channels handling incoming SMTP over TCP/IP mail, usually `tcp_local` and `tcp_intranet`. After doing so, review the most recent `tcp_local_slave.log-`*uniqueid* files to identify any particular characteristics of the messages that time out. For example, if incoming messages with large numbers of recipients are timing out, consider using the `expandlimit` keyword on the channel.
   Remember that if your system is overloaded and overextended, timeouts will be difficult to avoid entirely.

Messages Are Not Dequeued

Errors encountered during TCP/IP delivery are often transient; the MTA will generally retain messages when problems are encountered and retry them periodically. It is normal on large networks to experience periodic outages on certain hosts while other host connections work fine. To verify the problem, examine the log files for errors relating to delivery attempts. You may see error messages such as, "Fatal error from smtp_open." Such errors are not uncommon and are usually associated with a transient network problem. To debug TCP/IP network problems, use utilities like PING, TRACEROUTE, and NSLOOKUP.

The following example shows the steps you might use to see why a message is sitting in the queue awaiting delivery to `xtel.co.uk`. To determine why the message is not being dequeued, you can recreate the steps the MTA uses to deliver SMTP mail on TCP/IP.

```
% nslookup -query=mx xtel.co.uk (Step 1)

Server: LOCALHOST
Address: 127.0.0.1

Non-authoritative answer:
XTEL.CO.UK  preference = 10, mail exchanger = nsfnet-relay.ac.uk (Step 2)

% telnet nsfnet-relay.ac.uk 25 (Step 3)
Trying... [128.86.8.6]
telnet: Unable to connect to remote host: Connection refused
```

1. Use the NSLOOKUP utility to see what MX records, if any, exist for this host. If no MX records exist, then you should try connecting directly to the host. If MX records do exist, then you must connect to the designated MX relays. The MTA honors MX information preferentially, unless explicitly configured not to do so. See also TCP/IP MX Record Support.
2. In this example, the DNS (Domain Name Service) returned the name of the designated MX relay for `xtel.co.uk`. This is the host to which the MTA will actually connect. If more than one MX relay is listed, the MTA will try each MX record in succession, with the lowest preference value tried first.
3. If you do have connectivity to the remote host, you should check if it is accepting inbound SMTP connections by using TELNET to the SMTP server port 25.

> **Note**
> If you use TELNET without specifying the port, you will discover that the remote host accepts normal TELNET connections. This does not indicate that it accepts SMTP connections; many systems accept regular TELNET connections but refuse SMTP connections and vice versa. Consequently, you should always do your testing against the SMTP port.

In the previous example, the remote host is refusing connections to the SMTP port. This is why the MTA fails to deliver the message. The connection may be refused due to a misconfiguration of the remote host or some sort of resource exhaustion on the remote host. In this case, nothing can be done to locally to resolve the problem. Typically, you should let the MTA continue to retry the message.

If you are running Messaging Server on a TCP/IP network that does not use DNS, you can skip the first two steps. Instead, you can use TELNET to directly access the host in question. Be careful to use the same host name that the MTA would use. Look at the relevant log file from the MTA's last attempt to determine the host name. If you are using host files, you should make sure that the host name information is correct. It is strongly recommended that you use DNS instead of host names.

If you test connectivity to a TCP/IP host and encounter no problems using interactive tests, it is quite likely that the problem has simply been resolved since the MTA last tried to deliver the message. You can re-run the `imsimta submit tcp_channel` on the appropriate channel to see if messages are being dequeued.

To Create a New Channel

In certain circumstances, a remote domain can break down and the volume of mail addressed to this server can be so great that the outgoing channel queue fills up with messages that cannot be delivered. The MTA tries to redeliver these messages periodically (the frequency and number of the retries is configurable using the `backoff` keywords) and under normal circumstances, no action is needed. However, if too many messages get stuck in the queue, other messages may not get delivered in a timely manner because all the channel jobs are working to process the backlog of messages that cannot be delivered.

In this situation, you can reroute these messages to a new channel running in its own job controller pool. This will avoid contention for processing and allow the other channels to deliver their messages. This procedure is described in the following procedure. Assume a domain called `siroe.com`.

To Create a New Channel

1. Create a new channel called `tcp_siroe-daemon` and add a new value for the `pool` keyword. Channels are created in the channel block section of *msg-svr-base*`/config/imta.cnf`. The channel should have the same channel keywords on your regular outgoing `tcp_*` channel. Typically, this is the `tcp_local` channel, which handles all outbound (internet) traffic. Since `siroe.com` is out on the Internet, this is the channel to emulate. The new channel might look something like this:

```
tcp_siroe smtp nomx single_sys remotehost inner allowswitchchannel
\
dentnonenumeric subdirs 20 maxjobs 7 _pool SMTP_SIROE_ maytlsserver
\
maysaslserver saslswitchchannel tcp_auth missingrecipientpolicy 0
\
tcp_siroe-daemon
```

Note the new keyword-value pair pool SMTP_SIROE. This specifies that messages to this channel will only use computer resources from the SMTP_SIROE pool. A blank line is required before and after the new channel.

2. Add two rewrite rules to the rewrite rule section of the imta.cnf file to direct email destined for siroe.com to the new channel.
   The new rewrite rules look like this:

```
siroe.com     $U%$D@tcp_siroe-daemon
.siroe.com      $U%$H$D@tcp_siroe-daemon
```

   These rewrite rules direct messages to siroe.com (including addresses like host1.siroe.com or hostA.host1.siroe.com) to the new channel whose official host name is tcp_siroe-daemon. The rewriting part of these rules, $U%$D and $U%$H$D, retain the original addresses of the messages. $U copies the user name from original address. % is the separator—the @ between the username and domain. $H copies the unmatched portion of host/domain specification at the left of dot in pattern. $D copies the portion of domain specification that matched.

3. Define a new job controller pool called SMTP_SIROE.
   In *msg-svr-base*/config/job_controller.cnf add the following:

```
[POOL=SMTP_SIROE]
job_limit=10
```

   This creates a message resource pool called SMTP_SIROE that allows up to 10 jobs to be simultaneously run. Be sure not to leave any blank lines between this pool definition and the others. See The Job Controller for details on jobs and pools.

4. Restart the MTA.
   Issue the commands: imsimta cnbuild;imsimta restart
   This recompiles the configuration and restarts the job controller and dispatcher.
   In this example, a large quantity of email from your internal users is destined for a particular remote site called siroe.com. For some reason, siroe.com, is temporarily unable to accept incoming SMTP connections and thus cannot deliver email. (This type of situation is not a rare occurence.)
   As email destined for siroe.com comes in, the outgoing channel queue, typically tcp_local, will fill up with messages that cannot be delivered. The MTA tries to redeliver these messages periodically (the frequency and number of the retries is configurable using the backoff keywords) and under normal circumstances, no action is needed.
   However, if too many messages get stuck in the queue, other messages may not get delivered in a timely manner because all the channel jobs are working to process the backlog of siroe.com messages. In this situation, you may wish reroute siroe.com messages to a new channel running in its own job controller pool (see The Job Controller). This will allow the other channels to deliver their messages without having to contend for processing resources used by siroe.com messages. Creating a new channel to address this situation is described in the following information.

MTA Messages Are Not Delivered

In addition to message transport problems, there are two common problems which can result in unprocessed messages in the message queues:

1. The queue cache is not synchronized with the messages in the queue directories. Message files in

the MTA queue subdirectories that are awaiting delivery are entered into an in-memory queue cache. When channel programs run, they consult this queue cache to determine which messages to deliver in their queues. There are circumstances where there are message files in the queue, but there is no corresponding queue cache entry.

a. To check if a particular file is in the queue cache, you can use the `imsimta cache -view` utility. If the file is not in the queue cache, then the queue cache needs to be synchronized.
The queue cache is normally synchronized every four hours. If required, you can manually resynchronize the cache by using the command `imsimta cache -sync`. Once synchronized, the channel programs will process the originally unprocessed messages after new messages are processed. If you want to change the default (4 hours), you should modify the `job_controller.cnf` file in directory *msg-svr-base*`/config` by adding `sync_time=`*timeperiod* where *timeperiod* reflects how often the queue cache is synchronized. The *timeperiod* must be greater than 30 minutes. In the following example, the queue cache synchronization is modified to 2 hours by adding the `sync_time=02:00` to the global defaults section of the `job_controller.cnf`:

```
! VERSION=5.0
!IMTA job controller configuration file
!
!Global defaults
tcp_port=27442
secret=N1Y9[HzQKW
slave_command=NULL
sync_time=02:00
```

You can run `imsimta submit` *channel* to clear out the backlog of messages after running `imsimta cache -sync`. Clearing out the channel may take a long time if the backlog of messages is large (greater than 1000).
For summarized queue cache information, run `imsimta qm -maint dir -database -total`.

b. If after synchronizing the queue cache, messages are still not being delivered, you should restart the Job Controller. To do so, use the `imsimta restart job_controller` command.
Restarting the Job Controller causes the message data structure to be rebuilt from the message queues on disk.

> **Caution**
> Restarting the Job Controller is a drastic step and should only be performed after all other avenues have been thoroughly exhausted.

Refer to The Job Controller for more information on the Job Controller.

2. Channel processing programs fail to run because they cannot create their processing log file. Check the access permissions, disk space and quotas.

Messages are Looping

If the MTA detects that a message is looping, that message will be sidelined as a .HELD file. See Diagnosing and Cleaning up .HELD Messages. Certain cases can lead to message loops which the MTA can not detect.

The first step is to determine why the messages are looping. You should look at a copy of the problem message file while it is in the MTA queue area, MTA mail log entries (if you have the `logging` channel

keyword enabled in your MTA configuration file for the channels in question) relating to the problem message, and MTA channel debug log files for the channels in question. Determining the From: and To: addresses for the problem message, seeing the Received: header lines, and seeing the message structure (type of encapsulation of the message contents), can all help pinpoint which sort of message loop case you are encountering.

Some of the more common cases include:

1. A postmaster address is broken.
   The MTA requires that the postmaster address be a functioning address that can receive email. If a message to the postmaster is looping, check that your configuration has a proper postmaster address pointing to an account that can receive messages.
2. Stripping of Received: header lines is preventing the MTA from detecting the message loop.
   Normal detection of message loops is based on Received: header lines. If Received: header lines are being stripped (either explicitly on the MTA system itself, or on another system like a firewall), it can interfere with proper detection of message loops. In these scenarios, check that no undesired stripping of Received: header lines is occurring. Also, check for the underlying reason why the messages are looping. Possible reasons include: a problem in the assignment of system names or a system not configured to recognize a variant of its own name, a DNS problem, a lack of authoritative addressing information on the system in question, or a user address forwarding error.
3. Incorrect handling of notification messages by other messaging systems are generating reencapsulated messages in response to notification messages.
   Internet standards require that notification messages (reports of messages being delivered, or messages bouncing) have an empty envelope From: address to prevent message loops. However, some messaging systems do not correctly handle such notification messages. When forwarding or bouncing notification messages, these messaging systems may insert a new envelope From: address. This can then lead to message loops. The solution is to fix the messaging system that is incorrectly handling the notification messages.

Diagnosing and Cleaning up .HELD Messages

If the MTA detects a serious problem having to do with delivery of a message, the message is stored in a file with the suffix `.HELD` in *msg-svr-base*`/data/queue/`*channel*. For example:

```
% ls
ZZ0HXZ00G0EBRBCP.HELD
ZZ0HY200C0O6LGHU.HELD
ZZ0HYA006LP66O3H.HELD
ZZ0HZ7003EOQSE37.HELD
```

`.HELD` files can occur due to three major reasons:

- Looping messages. The MTA detected that the messages were looping via build-up of one or another sort of `Received:` header lines).
- User or domain status set to `hold`. These are messages that are, by intent of the MTA administrator, intentionally being side-lined, typically while some maintenance procedure is being performed, (for example, while moving user mailboxes).
- Suspicious messages. Messages that met some suspicion thresh hold and were held for later manual inspection by the MTA administrator. Messages can be `.HELD` due to exceeding a configured maximum number of envelope recipients (see the `holdlimit` channel keyword in Configuring Channel Definitions), due to running the `imsimta qclean`, `clean`, or `hold` commands based on some suspicion of the message(s) in question, or due to use of a `hold` action in a Sieve script.

Messages .HELD Due to Looping

Messages bouncing between servers or channels are said to be looping. Typically, a message loop occurs because each server or channel thinks the other is responsible for delivery of the message. Looping messages usually have a great many `*Received:` header lines. The `Received:` header lines illustrate the exact path of the message loop. Look carefully at the host names and any recipient address information (for example, `for recipient` clauses or `ORCPT recipient` comments) appearing in such header lines. One cause of such message loops is user error.

For example, an end users might set an option to forward messages on two separate mail hosts to one another. On their `sesta.com` account, the users enable mail forwarding to their `varrius.com` account. And, forgetting that they have enabled this setting, they set mail forwarding on their `varrius.com` account to their `sesta.com` account.

A loop can also occur with a faulty MTA configuration. For example, MTA Host X thinks that messages for `mail.sesta.com` go to Host Y. However, Host Y thinks that Host X should handle messages for `mail.sesta.com`. As a result, Host Y returns the mail to Host X.

In these cases, the message is ignored by the MTA and no further delivery is attempted. When such a problem occurs, look at the header lines in the message to determine which server or channel is bouncing the message. Fix the entry as needed.

Another common cause of message loops is the MTA receiving a message that was addressed to the MTA host using a network name that the MTA does not recognize (has not been configured to recognize) as one of its own names. The solution is to add the additional name to the list of names that your MTA recognizes as *its own*. The MTA's threshholds for determining that a message is looping are configurable; see the `MAX_*RECEIVED_LINES option.dat` options (http://download.oracle.com/docs/cd/E19566-01/819-4429/index.html). Also note that the MTA may optionally be configured. See the `HELD_SNDOPR` global MTA option to generate a syslog notice whenever a message is forced into `.HELD` state due to exceeding such a threshhold. If syslog messages of `Received count exceeded; message held.` are present, then you know that this is occurring.

You can resend the `.HELD` message by using the `imsimta qm release`, or by following these steps:

> **Note**
> The `imsimta qm release` is the preferred method.

1. Rename the `.HELD` extension to any 2 digit number other than 00. For example, .HELD to .06.

   > **Note**
   > Before renaming the .HELD file, be sure that the message has stopped looping.

2. Run `imsimta cache -sync`.
   Running this command updates the cache.
3. Run `imsimta submit` *channel* or `imsimta run` *channel*.

You might need to perform these steps multiple times, since the message may again be marked as `.HELD`, because the `Received:` header lines accumulate. If the problem still exists, the `*.HELD` file is recreated under the same channel with as before. If the problem has been addressed, the messages are dequeued and delivered.

If you determine that the messages can simply be deleted with no attempt to deliver them, see `clean` in *Messaging Server Administration Reference*.

Messages .HELD Due to User or Domain `hold` Status

Messages that are `.HELD` due to a user or domain status of `hold`--and only messages `.HELD` for such a reason--are normally stored in the hold channel's queue area. That is, `.HELD` message files in the hold

channel's queue area can be assumed to be `.HELD` due to user or domain status.

Messages .HELD Due to a Suspicious Characteristic

Messages `.HELD` due to some suspicious characteristic exhibit that characteristic. The characteristic could be anything that the site has chosen to characterize as *suspicious*. MTA Administrators should stay aware of these configuration choices and actions. However, if you are not the only or original administrator of this MTA, then check the MTA configuration for any configured use of the `holdlimit` channel keyword (Expansion of Multiple Addresses), any use of the `$H` flag in address-based `*_ACCESS` mapping tables in the MTA mappings file, or any use of the `hold` action in any system Sieve file (the system level `imta.filter` file, or any channel level Sieve filters configured and named by use of `sourcefilter` or `destinationfilter` channel keywords. See Specifying Mailbox Filter File Location . Additionally, ask any fellow MTA administrators about any manual command-line message holds (through, for instance, an `imsimta qm clean` command) they might have recently performed. Application of a Sieve filter `hold` action, whether from a system Sieve filter or from users' personal Sieve filters, may optionally be logged. See the `LOG_FILTER` global MTA option (http://download.oracle.com/docs/cd/E19566-01/819-4429/index.html)for more information.

Received Message is Encoded

Messages sent by the MTA are received in an encoded format. For example:

```
Date: Wed, 04 Jul 2001 11:59:56 -0700 (PDT)
From: "Desdemona Vilalobos" <Desdemona@sesta.com>
To: santosh@varrius.com
Subject: test message with 8bit data
MIME-Version: 1.0
Content-type: TEXT/PLAIN; CHARSET=ISO-8859-1
Content-transfer-encoding: QUOTED-PRINTABLE

2=00So are the Bo=F6tes Void and the Coal Sack the same?=
```

These messages appear unencoded when read with the MTA decoder command `imsimta decode`. See *Messaging Server Administration Reference* for more information.

The SMTP protocol only allows the transmission of ASCII characters (a seven-bit character set) as set forth by RFC 821. In fact, the unnegotiated transmission of eight-bit characters is illegal through SMTP, and it is known to cause a variety of problems with some SMTP servers. For example, SMTP servers can go into compute bound loops. Messages are sent over and over again. Eight-bit characters can crash SMTP servers. Finally, eight-bit character sets can wreak havoc with browsers and mailboxes that cannot handle eight-bit data.

An SMTP client used to only have three options when handling a message containing eight-bit data: return the message to the sender as undeliverable, encode the message, or send it in direct violation of RFC 821. But with the advent of MIME and the SMTP extensions, standard encodings exist that can encode eight-bit data by using the ASCII character set.

In the previous example, the recipient received an encoded message with a MIME content type of TEXT/PLAIN. The remote SMTP server (to which the MTA SMTP client transferred the message) did not support the transfer of eight-bit data. Because the original message contained eight-bit characters, the MTA had to encode the message.

Server-Side Rules (SSR) Are Not Working

A filter consists of one or more conditional actions to apply to a mail message. Since the filters are stored and evaluated on the server, they are often referred to as server-side rules (SSR).

This section includes information on the following SSR topics:

- Testing Your SSR Rules
- Common Syntax Problems

See also To Debug User-level Filters.

Testing Your SSR Rules

- To check the MTA's user filters, run the following command:

```
# imsimta test -rewrite -debug -filter <user>@<domain>
```

  In the output, look for the following information:

```
mmc_open_url called to open ssrf: <user>@ims-ms
   URL with quotes stripped: ssrd: <user>@ims-ms
Determined to be a SSRD URL.
   Identifier: <user>@ims-ms-daemon
Filter successfully obtained.
```

- In addition, you can add the `slave_debug` keyword to the `tcp_local` channel to see how a filter is applied. The results are displayed in the `tcp_local_slave.log` file. Be sure to add `mm_debug=5` in the `option.dat` file in the *msg-svr-base*`/config` directory to get sufficient debugging information.

Common Syntax Problems

- If there is a syntax problem with the filter, look for the following message in the `tcp_local_slave.log-*` file:
  `Error parsing filter expression:...`
  - If the filter is good, then filter information is at the end of the output.
  - If the filter is bad, then the following error is at the end of the output:
    `Address list error – 4.7.1 Filter syntax error: desdaemona@sesta.com`
    Also, if the filter is bad, then the SMTP RCPT TO command returns a temporary error response code:

```
RCPT TO: <user>@<domain>
452 4.7.1 Filter syntax error
```

Slow Response After Users Press Send Email Button

If users are experiencing delays when they send messages, undersized message queue disks could be responsible for reduced disk input/output. When users press the SEND button on their email client, the MTA will not fully accept receipt of the message until the message has been committed to the message queue. See Disk Sizing for MTA Message Queues for more information.

Asterisks in the Local Parts of Addresses or Received Fields

The MTA now checks for 8-bit characters (instead of just ASCII characters) in the local parts of addresses as well as the received fields it constructs and replaces them with asterisks.

Abnormal Job Controller Terminations Seen in `job_controller` Logs

The Job Controller is essentially an in-memory database. Unlike other parts of the MTA, it doesn't have queues or transactions with which to contend. It listens for activity coming in on various network connections and updates its database accordingly.

Consequently, if the Job Controller fails, it is most likely a resource allocation failure (resource exhaustion). The only significant resource the Job Controller uses, especially when under stress, is memory. Therefore, allocate the right amount of memory for the machine that contains the Job Controller. See the topic on planning a Messaging Server sizing strategy in *Unified Communications Suite Deployment Planning Guide* for details on memory utilization.

General Error Messages

When the MTA fails to start, general error messages appear at the command line. In this section, common general error messages will be described and diagnosed.

> **Note**
> To diagnose your own MTA configuration, use the `imsimta test -rewrite -debug` utility to examine your MTA's address rewriting and channel mapping process. This utility enables you to check the configuration without actually sending a message. See Check the MTA Configuration.

MTA subcomponents might also issue other error messages that are described in the MTA command-line utilities and configuration information in *Messaging Server Administration Reference*, Configuring POP, IMAP, and HTTP Services, and About MTA Services and Configuration. This section includes the following types of errors:

- Errors in mm_init
- Compiled Configuration Version Mismatch
- Swap Space Errors
- File Open or Create Errors
- Illegal Host/Domain Errors
- Errors in SMTP channels, os_smtp_* errors

Errors in mm_init

An error in `mm_init` generally indicates an MTA configuration problem. If you run the `imsimta test -rewrite` utility, these errors are displayed. Other utilities such as `imsimta cnbuild`, or a channel, a server, or a browser might also return such an error.

Commonly encountered mm_init errors include:

- bad equivalence for alias. . .
- cannot open alias include file. . .
- duplicate aliases found. . .
- duplicate host in channel table. . .
- duplicate mapping name found. . .
- mapping name is too long. . .
- error initializing ch_ facility compiled character set version mismatch
- error initializing ch_ facility no room in. . .
- local host alias or proper name too long for system. . .
- no equivalence addresses for alias. . .
- no official host name for channel. . .
- official host name is too long

bad equivalence for alias. . .

The right-hand side of an alias file entry is improperly formatted.

cannot open alias include file. . .

A file included into the alias file cannot be opened.

duplicate aliases found. . .

Two alias file entries have the same left hand side. You need to find and eliminate the duplication. Look for an error message that says `error line #XXX` where XXX is a line number. You can fix the duplicated alias on the line.

duplicate host in channel table. . .

This error message indicates that you have two channel definitions in the MTA configuration that both have the same official host name.

Note that an extraneous blank line in the rewrite rules (upper portion) of your MTA configuration file (`imta.cnf`) causes the MTA to interpret the remainder of the configuration file as channel definitions. Make sure that the very first line of the file is not a blank. Since there are often multiple rewrite rules with the same pattern (left-hand side), this then causes MTA to interpret them as channel definitions with non-unique official host names. Check your MTA configuration for any channel definitions with duplicate official host names and for any improper blank lines in the upper (rewrite rules) portion of the file.

duplicate mapping name found. . .

This message indicates that two mapping tables have the same name, and one of the duplicate mapping tables needs to be removed. However, formatting errors in the mapping file might cause the MTA to wrongly interpret something as a mapping table name. For example, failure to properly indent a mapping table entry causes the MTA to think that the left-hand side of the entry is actually a mapping table name. Check your mapping file for general form and check the mapping table names.

> **Note**
> A blank line should precede and follow any line with a mapping table name. However, no blank lines should be interspersed among the entries of a mapping table.

mapping name is too long. . .

This error means that a mapping table name is too long and needs to be shortened. Formatting errors in the mapping file might cause the MTA to wrongly interpret something as a mapping table name. For example, failure to properly indent a mapping table entry causes the MTA to think that the left-hand side of the entry is actually a mapping table name. Check your mapping file and mapping table names.

error initializing ch_ facility compiled character set version mismatch

If you see this message, you need to recompile and reinstall your compiled character set tables through the command `imsimta chbuild`. See `imsimta chbuild` in *Messaging Server Administration Reference* for more information.

error initializing ch_ facility no room in. . .

This error message generally means that you need to resize your MTA character set internal tables and then rebuild the compiled character set tables with the following commands:

```
imsimta chbuild -noimage -maximum -option
imsimta chbuild
```

Verify that nothing else needs to be recompiled or restarted before making this change. See `imsimta chbuild` in *Messaging Server Administration Reference* for more information.

local host alias or proper name too long for system. . .

This error indicates that a local host alias or proper name is too long (the optional right hand side in the second or subsequent names in a channel block). However, certain syntax errors earlier in the MTA configuration file (an extraneous blank line in the rewrite rules, for instance) may cause MTA to wrongly interpret something as a channel definition. Aside from checking the indicated line of the configuration file, also check above that line for other syntax errors. In particular, if the line in which MTA issues this error is intended as a rewrite rule, then be sure to check for extraneous blank lines above it.

no equivalence addresses for alias. . .

An entry in the alias file is missing a right hand side (translation value).

no official host name for channel. . .

This error indicates that a channel definition block is missing the required second line (the official host name line). See the MTA configuration and command-line utilities information in *Messaging Server Administration Reference* and Configuring Channel Definitions for more information on channel definition blocks. A blank line is required before and after each channel definition block, but a blank line must not be present between the channel name and official host name lines of the channel definition. Blank lines are not permitted in the rewrite rules portion of the MTA configuration file.

official host name is too long

The official host name for a channel (second line of the channel definition block) is limited to 128 octets in length. If you are trying to use a longer official host name on a channel, shorten it to a place holder name, and then use a rewrite rule to match the longer name to the short official host name. You might see this scenario if you work with the `l` (local) channel host name. For example:

```
<Original l Channel:>
!delivery channel to local /var/mail store
l subdirs 20 viaaliasrequired maxjobs 7 pool LOCAL_POOL
walleroo.pocofronitas.thisnameismuchtoolongandreallymakesnosensebutitisan e
Place Holder:>
!delivery channel to local /var/mail store
l subdirs 20 viaaliasrequired maxjobs 7 pool LOCAL_POOL
newt

<Create Rewrite Rule:>
newt.salamander.lizard.gecko.komododragon.com   $U%$D@newt
```

When using the `l` (local) channel, you need to use a REVERSE mapping table. See the MTA configuration information in *Messaging Server Administration Reference* for information on usage and syntax.

Certain syntax errors earlier in the MTA configuration file (for example, an extraneous blank line in the rewrite rules) can cause the MTA to wrongly interpret something as a channel definition. This could result

in an intended rewrite rule being interpreted as an official host name. Besides checking the indicated line of the configuration file, also check above that line for other syntax errors. In particular, if the line on which the MTA issues this error is intended as a rewrite rule, be sure to check for extraneous blank lines above it.

Compiled Configuration Version Mismatch

One of the functions of the `imsimta cnbuild` utility is to compile MTA configuration information into an image that can be quickly loaded. The compiled format is quite rigidly defined and often changes substantially between different versions of the MTA. Minor changes might occur as part of patch releases.

When such changes occur, an internal version field is also changed so that incompatible formats can be detected. The MTA components halt with the "Compiled Configuration Version Mismatch" error when an incompatible format is detected. The solution to this problem is to generate a new, compiled configuration with the command `imsimta cnbuild.`

Also, use the `imsimta restart` command to restart any resident MTA server processes, so they can obtain updated configuration information.

Swap Space Errors

To ensure proper operation, it is important to configure enough swap space on your messaging system. The amount of required swap space will vary depending on your configuration. A general tuning recommendation is that the amount of swap space should be at least three times the amount of main memory.

An error message such as the following indicates a lack of swap space:

```
jbc_channels: chan_execute [1]: fork failed: Not enough space
```

You might see this error in the Job Controller log file. Other swap space errors will vary depending on your configuration.

Use the following commands to determine how much swap space you have left and determine how much you have used:

- Oracle Solaris systems: `swap -s` (at the time MTA processes are busy), `ps -elf`, or `tail /var/adm/messages`
- HP-UX systems: `swapinfo` or `tail /var/adm/syslog/syslog.log`

File Open or Create Errors

To send a message, the MTA reads configuration files and creates message files in the MTA message queue directories. Configuration files must be readable by the MTA or any program written against the MTA's SDKs. During installation, proper permissions are assigned to these files. The MTA utilities and procedures which create configuration files also assign permissions. If the files are protected by the system manager, other privileged user, or through some site-specific procedure, the MTA may not be able to read configuration information. This results in "File open&" errors or unpredictable behavior. The `imsimta test -rewrite` utility reports additional information when it encounters problems reading configuration files. See `imsimta test` in *Messaging Server Administration Reference*.

If the MTA appears to function from privileged accounts but not from unprivileged accounts, then file permissions in the MTA table directory are likely the cause of the problem. Check the permissions on configuration files and their directories. See Check the Ownership of Critical Files.

"File create" errors usually indicate a problem while creating a message file in an MTA message queue directory. See Check the Message Queue Directories to diagnose file creation problems.

Illegal Host/Domain Errors

You might see this error when an address is provided to the MTA through a browser. Or, the error may be deferred and returned as part of an error return mail message. In both cases, this error message indicates that the MTA is not able to deliver mail to the specified host. To determine why the mail is not being sent to the specified host, follow these troubleshooting procedures:

- Verify that the address in question is not misspelled, is not transcribed incorrectly, or does not use the name of a host or domain that no longer exists.
- Run the address in question through the `imsimta test -rewrite` utility. If this utility also returns an "illegal host/domain" error on the address, then the MTA has no rules in the `imta.cnf` file and related files to handle the address. Verify that you have configured MTA correctly, that you answered all configuration questions appropriately, and that you have kept your configuration information up to date.
- If `imsimta test -rewrite` does not encounter an error on the address, then MTA is able to determine how to handle the address, but the network transport will not accept it. You can examine the appropriate log files from the delivery attempt for additional details. Transient network routing or name service errors should not result in returned error messages, though it is possible for badly misconfigured domain name servers to cause these problems.
- If you are on the Internet, check that you have properly configured your TCP/IP channel to support MX record lookups. Many domain addresses are not directly accessible on the Internet and require that your mail system correctly resolve MX entries. If you are on the Internet and your TCP/IP is configured to support MX records, you should have configured the MTA to enable MX support. See TCP/IP Connection and DNS Lookup Support Configuring Channel Definitions for more information. If your TCP/IP package is not configured to support MX record lookups, then you will not be able to reach MX-only domains.

Errors in SMTP channels, os_smtp_* errors

Errors such as the following are not necessarily MTA errors: os_smtp_* errors like os_smtp_open, os_smtp_read, and os_smtp_write errors. These errors are generated when the MTA reports a problem encountered at the network layer. For example, an os_smtp_open error means that the network connection to the remote side could not be opened. The MTA may be configured to connect to an invalid system because of addressing errors or channel configuration errors. The os_smtp_* errors are commonly due to DNS or network connectivity problems, particularly if this was a previously working channel or address. os_smtp_read or os_smtp_write errors are usually an indication that the connection was aborted by the other side or due to network problems.

Network and DNS problems are often transient in nature. The occasional os_smtp_* error is usually nothing to be concerned about. However, if you are consistently seeing these errors, it could be an indication of an underlying network problem.

To obtain more information about a particular os_smtp_* error, enable debugging on the channel in question. Investigate the debug channel log file that will show details of the attempted SMTP dialogue. In particular, look at the timing of when a network problem occurred during the SMTP dialogue. The timing could suggest the type of network or remote side issue. In some cases, you might also want to perform network level debugging (for example, TCP/IP packet tracing) to determine what was sent or received.

Chapter 31. Monitoring Messaging Server

Monitoring Messaging Server

In most cases, a well-planned, well-configured server will perform without extensive intervention from an administrator. As an administrator, however, it is your job to monitor the server for signs of problems.

Topics:

See also:

Automatic Monitoring and Restart

Messaging Server provides a way to transparently monitor services and automatically restart them if they crash or become unresponsive (the services hangs or freeze up). It can monitor all message store, MTA, and MMP services including the IMAP, POP, HTTP, job controller, dispatcher, and MMP servers. It does not monitor other services such as SMS or TCP/SNMP servers. (TCP/SNMP is monitored by the job controller.) Refer to Automatic Restart of Failed or Unresponsive Services and Monitoring Using msprobe and watcher Functions.

Daily Monitoring Tasks

The most important tasks you should perform on a daily basis are checking postmaster mail, monitoring the log files, and setting up the `stored` utility. These tasks are described below.

Checking Postmaster Mail

Messaging Server has a predefined administrative mailing list set up for postmaster email. Any users who are part of this mailing list will automatically receive mail addressed to postmaster.

The rules for postmaster mail are defined in RFC822, which requires every email site to accept mail addressed to a user or mailing list named postmaster and that mail sent to this address be delivered to a real person. All messages sent to postmaster@host.domain are sent to a postmaster account or mailing list.

Typically, the postmaster address is where users should send email about their mail service. As postmaster, you might receive mail from local users about server response time, from other server administrators who are encountering problems sending mail to your server, and so on. You should check postmaster mail daily.

You can also configure the server to send certain error messages to the postmaster address. For example, when the MTA cannot route or deliver a message, you can be notified via email sent to the postmaster address. You can also send exception condition warnings (low disk space, poor server response) to postmaster.

Monitoring and Maintaining the Log Files

Messaging Server creates a separate set of log files for each of the major protocols or services it supports including SMTP, IMAP, POP, and HTTP. These are located in *msg-svr-base*`/data/log`. You should monitor the log files on a routine basis--especially if you are having problems with the server.

Be aware that logging can impact server performance. The more verbose the logging you specify, the more disk space your log files will occupy for a given amount of time. You should define effective but realistic log rotation, expiration, and backup policies for your server. For information about defining logging policies for your server, see Managing Logging.

Setting Up the msprobe Utility

The `msprobe` utility automatically performs monitoring and restart functions. For further information see Monitoring Using msprobe and watcher Functions.

Utilities and Tools for Monitoring

The following tools are available in for monitoring:

- immonitor-access
- imcheck
- Log Files
- imsimta counters
- imsimta qm counters
- MTA Monitoring Using SNMP
- Monitoring Using msprobe and watcher Functions

immonitor-access

`immonitor-access` monitors the status of the following Messaging Server components/processes: Mail Delivery (SMTP server), Message Access and Store (POP and IMAP servers), Directory Service (LDAP server) and HTTP server. This utility measures the response times of the various services and the total round trip time taken to send and retrieve a message. The Directory Service is monitored by looking up a specified user in the directory and measuring the response time. Mail Delivery is monitored by sending a message (SMTP) and the Message Access and Store is monitored by retrieving it. Monitoring the HTTP server is limited to finding out whether or not it is up and running.

For complete instructions, refer to `immonitor-access`.

imcheck

Use `imcheck` to monitor database statistics including logs and transactions.

counterutil

This utility provides statistics acquired from different system counters. For details, see Gathering Message Store Counter Statistics by Using counterutil.

Log Files

Messaging server logs event records for SMTP, IMAP, POP, and HTTP. The policies for creating and managing the Messaging Server log files are customizable.

Since logging can affect the server performance, logging should be considered very carefully before the burden is put on the server. Refer to Managing Logging for more information.

`imsimta counters`

The MTA accumulates message traffic counters based upon the Mail Monitoring MIB, RFC 1566 for each of its active channels. The channel counters are intended to help indicate the trend and health of your email system. Channel counters are not designed to provide an accurate accounting of message traffic. For precise accounting, instead see MTA logging as discussed in Managing Logging.

The MTA channel counters are implemented using the lightest weight mechanisms available so that they cause as little impact as possible on actual operation. Channel counters do not try harder: if an attempt to map the section fails, no information is recorded. If one of the locks in the section cannot be obtained almost immediately, no information is recorded. When a system is shut down, the information contained in the in-memory section is lost forever.

The `imsimta counters -show` command provides MTA channel message statistics (see below). These counters need to be examined over time noting the minimum values seen. The minimums may actually be negative for some channels. A negative value means that there were messages queued for a channel at the time that its counters were zeroed (for example, the cluster-wide database of counters created). When those messages were dequeued, the associated counters for the channel were decremented and therefore leading to a negative minimum. For such a counter, the correct "absolute" value is the current value less the minimum value that counter has ever held since being initialized.

```
Channel          Messages    Recipients    Blocks
-------          --------    ----------    -------
tcp_local
   Received       29379       79714      982252                (1)
   Stored            61         113       -2004                (2)
   Delivered      29369       79723      983903 (29369 first time)  (3)
   Submitted      13698       13699       18261                (4)
   Attempted          0           0           0                (5)
   Rejected           1          10           0                (6)
   Failed           104         104        4681                (7)
   Queue time/count        16425/29440 = 0.56                  (8)
   Queue first time/count  16425/29440 = 0.56                  (9)
   Total In Assocs           297637
   Total Out Assocs           28306
```

1) `Received` is the number of messages enqueued to the channel named `tcp_local`. That is, the messages enqueued (E records in the `mail.log*` file) to the `tcp_local` channel by any other channel.

2) `Stored` is the number of messages stored in the channel queue to be delivered.

3) `Delivered` is the number of messages which have been processed (dequeued) by the channel `tcp_local`. (That is, D records in the `mail.log*` file.) A dequeue operation may either correspond to a successful delivery (that is, an enqueue to another channel), or to a dequeue due to the message being returned to the sender. This will generally correspond to the number `Received` minus the number `Stored`.

The MTA also keeps track of how many of the messages were dequeued upon first attempt; this number is shown in parentheses.

4) `Submitted` is the number of messages enqueued (E records in the `mail.log` file) by the channel `tcp_local` to any other channel.

5) `Attempted` is the number of messages which have experienced temporary problems in dequeuing, that is, Q or Z records in the `mail.log*` file.

6) `Rejected` is the number of attempted enqueues which have been rejected, that is, `J` records in the `mail.log*` file.

7) `Failed` is the number of attempted dequeues which have failed, that is, `R` records in the `mail.log*` file.

8) `Queue time/count` is the average time-spent-in-queue for the delivered messages. This includes both the messages delivered upon the first attempt, see (9), and the messages that required additional delivery attempts (hence typically spent noticeable time waiting fallow in the queue).

9) `Queue first time/count` is the average time-spent-in-queue for the messages delivered upon the first attempt.

Note that the number of messages submitted can be greater than the number delivered. This is often the case, since each message the channel dequeues (delivers) will result in at least one new message enqueued (submitted) but possibly more than one. For example, if a message has two recipients reached via different channels, then two enqueues will be required. Or if a message bounces, a copy will go back to the sender and another copy may be sent to the postmaster. Usually that will be two submissions (unless both are reached through the same channel).

More generally, the connection between `Submitted` and `Delivered` varies according to type of channel. For example, in the conversion channel, a message would be enqueued by some other arbitrary channel, and then the conversion channel would process that message and enqueue it to a third channel and mark the message as dequeued from its own queue. Each individual message takes a path:

```
elsewhere -> conversion E record Received
conversion -> elsewhere E record Submitted
conversion              D record Delivered
```

However, for a channel such as `tcp_local` which is not a "pass through," but rather has two separate pieces (slave and master), there is no connection between `Submitted` and `Delivered`. The `Submitted` counter has to do with the SMTP server portion of the `tcp_local` channel, whereas the `Delivered` counter has to do with the SMTP client portion of the `tcp_local` channel. Those are two completely separate programs, and the messages travelling through them may be completely separate.

Messages submitted to the SMTP server:
`tcp_local -> elsewhere E record Submitted`

Messages sent out to other SMTP hosts via the SMTP client:

```
elsewhere -> tcp_local E record Received
tcp_local              D record Delivered
```

Channel dequeues (delivers) will result in at least one new message enqueued (submitted) but possibly more than one. For example, if a message has two recipients reached via different channels, then two enqueues will be required. Or if a message bounces, a copy will go back to the sender and another copy may be sent to the postmaster. Usually that will be reached through the same channel.

`imsimta counters` Implementation

For performance reasons, a node running the MTA keeps a cache of channel counters in memory using a shared memory section. As processes on the node enqueue and dequeue messages, they update the counters in this in-memory cache. If the in-memory section does not exist when a channel runs, the section will be created automatically. (The `imta start` command also creates the in-memory section, if it does not exist.)

The command `imta counters -clear` or the `imta qm` command `counters clear` may be used to reset the counters to zero.

imsimta qm counters

The `imsimta qm counters` utility displays MTA channel queue message counters. You must be root or `mailsrv` to run this utility. The output fields are the same as those described in `imsimta counters` in *Messaging Server Administration Reference*.

Example:

```
# imsimta counters -create
# imsimta qm counters show
Channel                Messages   Recipients Blocks
---------------------- ---------- ---------- ----------
tcp_intranet
   Received              13077      13859     264616
   Stored                   92         91       -362
   Delivered             12985      13768     264978
   Submitted              2594       2594       3641
...
```

Every time you restart the MTA, you must run: **`# imsimta counters -create`**

imsimta qm summarize

The `imsimta qm summarize` utility displays a summary of the number of messages and their status in the MTA channel queues.

qm summarize modes

Like many of the `qm` subcommands, `summarize` has two modes of operation: The `-directory_tree` mode examines the message files in the MTA queue directories on disk. The `-database` mode queries the `job_controller` process's in-memory database structures. The directory mode creates a heavier load on the IO system and may not reflect what `job_controller` is actually working on, but it can be useful to know if there is a difference between the two. The job controller makes the decisions about which messages are tried next, so the database mode will be more useful.

```
# imsimta qm
qm.maint> sum -directory_tree
                          Channel  Messages  Size (Mb)
-------------------------------  --------  ---------
                      conversion         0        0.0
                            hold         0        0.0
               ims-ms (stopped)         2        0.0
                         process         0        0.0
                       reprocess         0        0.0
          tcp_intranet (stopped)         0        0.0
             tcp_local (stopped)         2        0.0
-------------------------------  --------  ---------
                           Totals         4        0.0
```

Notice that the `-database` mode breaks down the number messages into three catagories. **Active** messages are currently being tried by a worker process. **Pending** messages are ready to be tried by a worker as soon as thread/slot is available. **Delayed** messages have been tried before and are waiting for a specified time to be tried again as per the `backoff` option for that channel.

```
qm.maint> sum -database
                                      Total
Total
                          Channel  Messages =  Active + Pending + Delayed
Size (Mb)
-------------------------------  --------  --------  --------  --------
---------
                      conversion         0         0         0         0
0.0
                            hold         0         0         0         0
0.0
               ims-ms (stopped)         2         0         2         0
0.0
                               l         0         0         0         0
0.0
                         process         0         0         0         0
0.0
                       reprocess         0         0         0         0
0.0
          tcp_intranet (stopped)         0         0         0         0
0.0
             tcp_local (stopped)         2         0         2         0
0.0
-------------------------------  --------  --------  --------  --------
---------
                           Totals         4         0         4         0
0.0
```

Note: In these examples, some channels had been stopped using `imsimta qm stop <channel>` to provide some data to look at.

Held messages

A `.HELD` message file is a message which has encountered a loop or otherwise been *sidelined* and

requires administrative intervention for some reason. You can see such messages using the `-held` switch. Note that job_controller will have no knowledge of held messages, therefore the `-database` and `-held` switches are mutually exclusive. For more information about `.HELD` messages see Diagnosing and Cleaning up .HELD Messages.

```
qm.maint> sum -held -database
%QM-E-CMDERR, Conflicting parameters and/or qualifiers: (DATABASE AND HELD)

qm.maint> sum -held

Held      Held
                          Channel  Messages  Size (Mb)      Oldest Queued
Messages  Size (Mb)         Oldest Held
--------------------------------  --------  ---------  -----------------
--------  ---------  -----------------
                       conversion         0        0.0
0         0.0
                            hold         0        0.0
1         0.0   23 Apr, 21:35:16
                ims-ms (stopped)         2        0.0    6 Apr, 13:24:00
0         0.0
                          process         0        0.0
0         0.0
                        reprocess         0        0.0
0         0.0
           tcp_intranet (stopped)         0        0.0
0         0.0
              tcp_local (stopped)         2        0.0    5 May, 10:16:08
0         0.0
--------------------------------  --------  ---------  -----------------
--------  ---------  -----------------
                            Totals         4        0.0    6 Apr, 13:24:00
1         0.0   23 Apr, 21:35:16
```

Displaying Summary by Destination Host

The `-hosts` switch displays a breakdown of the messages in the queue by destination host for channels where that is meaningful. This information is stored in the `job_controller` process in-memory queue cache database. Therefore `-hosts` implies `-database`.

```
qm.maint> sum -hosts
                                    Total
Total
Channel                      Host  Messages =  Active + Pending + Delayed
Size (Mb)
-------------------------------  --------  --------  --------  --------
---------
conversion                                0         0         0         0
0.0
hold                                      0         0         0         0
0.0
ims-ms (stopped)                          2         0         2         0
0.0
l                                         0         0         0         0
0.0
process                                   0         0         0         0
0.0
reprocess                                 0         0         0         0
0.0
tcp_intranet (stopped)                    0         0         0         0
0.0
tcp_local (stopped)                       2         0         2         0
0.0
                          aol.com         1         0         1         0
0.0
                          sun.com         1         0         1         0
0.0
-------------------------------  --------  --------  --------  --------
---------
Totals                                    4         0         4         0
0.0
```

imsimta qm jobs

After starting the tcp_local channel:

```
tcp_local                                 1         1         0         0
0.0
                          aol.com         1         1         0         0
0.0
```

And to see what processes are working on what jobs:

```
qm.maint> jobs tcp_local
tcp_local channel:

   Pending: 0 jobs
   Active:  1 jobs, 1 messages (0.00 Mb), 1 recipients
   Current jobs have delivered 1 messages, requeued 0 messages

   Active jobs and messages:

       22157: 1 messages (0.00 Mb), 1 recipients
              1 messages processed and 0 requeued

              Active hosts:

                 aol.com

              Active messages:

                 ZZg0u410_P_~1.01 (1.0 Kb)
```

MTA Monitoring Using SNMP

Messaging Server supports system monitoring through the Simple Network Management Protocol (SNMP). Using an SNMP client (sometimes called a *network manager*) such as Sun Net Manager or HP OpenView (not provided with this product), you can monitor certain parts of the Messaging Server. Refer to SNMP Support for details.

Monitoring Using msprobe and watcher Functions

Messaging Server provides two processes, `watcher` and `msprobe` to monitor various system services. `watcher` watches for server crashes and restarts them as necessary. `msprobe` monitors server hangs (unresponsiveness). Specifically `msprobe` monitors the following:

- *Server Response Time*. `msprobe` connects to the enabled servers using their protocol commands and measures their response times. If the response time exceeds the alarm warning threshold, an alarm message is sent (see Alarm Messages to a server, or the server response time exceeds a specified timeout period, the server is restarted. Server response times are recorded in a counter database and is logged to the default log file. `counterutil` can be used to display the server response time statistics (counterutil).

The following servers are monitored by `msprobe`: `imap`, `pop`, `http`, `cert`, `job_controller`, `smtp`, `lmtp`, `mmp` and `ens`. When `smtp` or `lmtp` are not responding, the dispatcher is restarted. `ens` cannot be automatically restarted.

- *Disk usage*. `msprobe` checks the disk availability and usage for every message store partition. Specifically it checks the message store `mboxlist` database directory and the MTA queue directory. If disk usage exceeds a configured threshold, an alarm message is sent. The disk sizes and usages are recorded in a counter database and is logged to the default log file. Administrators can use the `counterutil` utility (see `counterutil`) to display the disk usage statistics.

- **Message Store mboxlist Database Log File Accumulation**. Log file accumulation is an indication of an `mboxlist` database error. `msprobe` counts the number of active log files and if the number of active log files is larger than the threshold, `msprobe` logs a critical error message to the `default` log file to inform the admin to restart the server. If the autorestart is enabled (`local.autorestart` to `yes`), the store daemon is restarted.

`watcher` and `msprobe` are controlled by the `configutil` options shown in the table below. Further information can be found in Automatic Restart of Failed or Unresponsive Services.

***msprobe* and *watcher configutil* Options**

Options	Description
`local.autorestart`	Enable automatic server restart. Automatically restarts failed or hung services. Default: no
`local.autorestart.timeout`	Failure retry time-out. If a server fails more than twice in this designated amount of time, then the system stops trying to restart the server. The value (set in seconds) should be set to a period value longer than the `msprobe` interval (`local.schedule.msprobe`). Default: 600 seconds
`local.probe.service.timeout`	Timeout for a specific server before restart. *service* can be imap, pop, http, cert, job_controller, smtp, lmtp, mmp or ens. Default: use `service.readtimeout`
`local.probe.service.warningthreshold`	Number of seconds of a specific server's non-response before a warning message is logged to `default` log file. *service* can be imap, pop, http, cert, job_controller, smtp, lmtp, mmp or ens. Default: Use `local.probe.warningthreshold`
`local.probe.warningthreshold`	Number of seconds of server non-response before a warning message is logged to `default` log file. Default: 25 secs
`local.queuedir`	MTA queue directory to check if queue size exceeded threshold defined by `alarm.diskavail.msgalarmthreshold`. Default: none
`service.readtimeout`	Period of server non-response before restarting that server. See `local.schedule.msprobe`. Default: 30 seconds
`local.schedule.msprobe`	`msprobe` run schedule. A `crontab` style schedule string (see `local.schedule.expire` in the Expire and Purge configutil Log and Scheduling Parameters Table). Note that by default, this is automatically set. See Pre-defined Automatic Tasks. To disable: set `local.schedule.msprobe.enable` to `NO`.
`local.watcher.enable`	Enable watcher which monitors service failures. (IMAP, POP, HTTP, job controller, dispatcher, message store (`stored`), `imsched`, and MMP. (LMTP/SMTP servers are monitored by the dispatcher and LMTP/SMTP clients are monitored by the job_controller.) Logs error messages to the default log file for specific failures. Default: on

Alarm Messages

`msprobe` can issue alarms in the form of email messages to the postmaster (see To Monitor imapd, popd and httpd) warning of a specified condition. A sample email alarm sent when a certain threshold is exceeded is shown below:

```
Subject:    ALARM: server response time in seconds of
"ldap_siroe.com_389" is 10
Date:    Tue, 17 Jul 2001 16:37:08 -0700 (PDT)
From:    postmaster@siroe.com
To:     postmaster@siroe.com
Server instance: /opt/SUNWmsgsr
Alarmid: serverresponse
Instance: ldap_siroe_europa.com_389
Description: server response time in seconds
Current measured value (17/Jul/2001:16:37:08 -0700): 10
Lowest recorded value: 0
Highest recorded value: 10
Monitoring interval: 600 seconds
Alarm condition is when over threshold of 10
Number of times over threshold: 1
```

You can specify how often `msprobe` monitors disk and server performance, and under what circumstances it sends alarms. This is done by using the `configutil` command to set the alarm parameters. The table below shows some useful alarm parameters along with their default setting. See http://msg.wikidoc.info/index.php/Configutil_Reference for a complete list of all parameters.

Useful Alarm Message `configutil` Parameters

Parameter	Description (Default in parenthesis)
alarm.msgalarmnoticehost	(localhost) Machine to which you send warning messages.
alarm.msgalarmnoticeport	(25) The SMTP port to which to connect when sending alarm message.
alarm.msgalarmnoticercpt	(Postmaster@localhost) Whom to send alarm notice.
alarm.msgalarmnoticesender	(Postmaster@localhost) Address of sender the alarm.
alarm.diskavail.msgalarmdescription	(Percentage mail partition diskspace available.) Text for description field for disk availability alarm.
alarm.diskavail.msgalarmstatinterval	(3600) Interval in seconds between disk availability checks. Set to 0 to disable checking of disk usage.
alarm.diskavail.msgalarmthreshold	(10) Percentage of disk space availability below which an alarm is sent.
alarm.diskavail.msgalarmthresholddirection	(-1) Specifies whether the alarm is issued when disk space availability goes below threshold (-1) or above it (1).
alarm.diskavail.msgalarmwarninginterval	(24). Interval in hours between subsequent repetition of disk availability alarms.
alarm.serverresponse.msgalarmdescription	(Server response time in seconds.) Text for description field for servers response alarm.
alarm.serverresponse.msgalarmstatinterval	(600) Interval in seconds between server response checks. Set to 0 to disable checking of server response.
alarm.serverresponse.msgalarmthreshold	(10) If server response time in seconds exceeds this value, alarm issued.
alarm.serverresponse.msgalarmthresholddirection	(1) Specifies whether alarm is issued when server response time is greater that (1) or less than (-1) the threshold.
alarm.serverresponse.msgalarmwarninginterval	(24) Interval in hours between subsequent repetition of server response alarm.

Monitoring LDAP Directory Server

Monitoring LDAP Directory Server

The LDAP directory server (`slapd`) provides directory information for the messaging system. If `slapd` is down, the system will not work properly. If `slapd` response time is too slow, this will affect login speed and any other transaction that requires LDAP lookups.

Topics:

Symptoms of slapd Problems

- Client POP, IMAP, or Webmail Authentication fails or slower than expected.
- MTA not working properly

To Monitor slapd

- Check that `ns-slapd` process is running.
- Check `slapd` log files `access` and `errors` in `slapd-`*instance* `/logs/`
- Check the `ns-slapd` response time while searching for a user.
- See also immonitor-access.

Monitoring System Performance

Monitoring System Performance

This information focuses on Messaging Server monitoring, however, you also need to monitor the system on which the server resides. A well-configured server cannot perform well on a poorly-tuned system, and symptoms of server failure may be an indication that the hardware is not powerful enough to serve the email load. This information does not provide all the details for monitoring system performance as many of these procedures are platform specific and may require that you refer to the platform specific system documentation.

Topics:

Monitoring End-to-end Message Delivery Times

Email needs to be delivered on time. This may be a service agreement requirement, but also it is good policy to have mail delivered as quickly as possible. Slow end-to-end times could indicate a number of things. It may be that the server is not working properly, or that certain times of the day experience overwhelming message loads, or that the existing hardware resources are being pushed beyond their capacity.

Symptoms of Poor End-to-end Message Delivery Times

Mail takes a longer period of time to be delivered than normal.

To Monitor End-to-end Message Delivery Times

- Use any facility that sends a message and receives it. Compare the headers times between server hops, and times between point of origin and retrieval. See immonitor-access.

Monitoring CPU Usage

High CPU usage is either a sign that there is not enough CPU capacity for the level of usage or some process is using up more CPU cycles than is appropriate.

Symptoms of CPU Usage Problems

Poor system response time. Slow logging in of users. Slow rate of delivery.

To Monitor CPU Usage

Monitoring CPU usage is a platform specific task. Refer to the relevant platform documentation.

Monitoring the MTA

Monitoring the MTA

Topics:

Monitoring the Size of the Message Queues

Excessive message queue growth may indicate that messages are not being delivered, are being delayed in their delivery, or are coming in faster than the system can deliver them. This may be caused by a number of reasons such as a denial of service attack caused by huge numbers of messages flooding your system, or the Job Controller not running.

See Channel Message Queues, Messages Are Not Dequeued, and MTA Messages Are Not Delivered for more information on message queues.

Symptoms of Message Queue Problems

- Disk space usage grows.
- User not receiving messages in a reasonable time.
- Message queue sizes are abnormally high.

To Monitor the Size of the Message Queues

Probably the best way to monitor the message queues is to use `imsimta qm counters` and `imsimta qm summarize`.

You can also monitor the number of files in the queue directories (*msg-svr-base*`/data/queue/`). The number of files will be site-specific, and you'll need to build a baseline history to find out what is "too many." This can be done by recording the size of the queue files over a two week period to get an approximate average.

Checking for Held messages

If the MTA detects a message is looping, it will be *sidelined* by renaming the queue message file to `.HELD`. For more discussion of how messages can become `.HELD` and what to do about them, refer to Diagnosing and Cleaning up .HELD Messages. To see whether there are any held messages, use `imsimta qm summarize -held`.

Monitoring Rate of Delivery Failure

A delivery failure is a failed attempt to deliver a message to an external site. A large increase in rate of delivery failure can be a sign of a network problem such as a dead DNS server or a remote server timing out on responding to connections.

Symptoms of Rate of Delivery Failure

There are no outward symptoms. Lots of `Q` records will appear in to `mail.log_current`.

To Monitor the Rate of Delivery Failure

Delivery failures are recorded in the MTA logs with the logging entry code `Q`. Look at the record in the file *msg-svr-base*`/data/log/mail.log_current`. Example:

```
mail.log:06-Oct-2003 00:24:03.66 501d.0b.9 ims-ms Q 5 durai.balusamy@Sun.COM
rfc822;durai.balusamy@Sun.COM durai@ims-ms-daemon
<00ce01c38bda$c7e2b240$6501a8c0@guindy>Mailbox is busy
```

Monitoring Inbound SMTP Connections

An unusual increase in the number of inbound SMTP connections from a given IP address may indicate:

- An external user is trying to relay mail.
- An external user is trying to do a service denial attack.

Symptoms of Unauthorized SMTP Connections

- **External user relaying mail** : No outward symptoms.
- **Service denial attack**: External attempt to overload the SMTP servers with message requests.

To Monitor Inbound SMTP Connections

- **External user relaying mail**: Look in *msg-svr-base*`/log/mail.log_current` for records with the logging entry code `J` (rejected relays). To turn on logging of remote IP addresses add the following line to the `option.dat` file:

  ```
  log_connection=1
  ```

  Note that there is a slight performance trade-off when this feature is enabled.
- **Service denial attack**: To find out who and how many users are connecting to the SMTP servers, you can run the command `netstat` and check for connections at the SMTP port (default: 25). Example:

```
Local address       Remote address                                State
192.18.79.44.25     192.18.78.44.56035    32768   0  32768   0   CLOSE_WAIT
192.18.79.44.25     192.18.136.54.57390    8760   0  24820   0   ESTABLISHED
192.18.79.44.25     192.18.26.165.48508   33580   0  24820   0   TIME_WAIT
```

Note that you will first need to determine the appropriate number of SMTP connections and their states (`ESTABLISHED`, `CLOSE_WAIT`, etc.) for your system to determine if a particular reading is out of the ordinary.

If you find many connections staying in the `SYN_RECEIVED` state this might be caused by a broken network or a denial of service attack. In addition, the lifetime of an SMTP server process is limited. This is controlled by the MTA configuration variable `MAX_LIFE_TIME` in the `dispatcher.cnf` file. The default is 86,400 seconds (one day). Similarly, `MAX_LIFE_CONNS` specifies the maximum number of connections a server process can handle in its lifetime. If you find a particular SMTP server that has around for a long time you may wish to investigate.

Monitoring the Dispatcher and Job Controller Processes

The Dispatcher and Job Controller Processes must be operating for MTA to work. You should have one process of each kind.

Symptoms of Dispatcher and Job Controller Processes Down

If the Dispatcher is down or does not have enough resources, SMTP connections are refused.

If the Job Controller is down, queue size will grow.

To Monitor Dispatcher and Job Controller Processes

Check to see that the processes called `dispatcher` and `job_controller` exist. See Check that the Job Controller and Dispatcher Are Running.

Chapter 32. SNMP Support

SNMP Support

The Messaging Server supports system monitoring through the Simple Network Management Protocol (SNMP). Using an SNMP client (sometimes called a *network manager*) such as Sun Net Manager or HP OpenView (not provided with the this product), you can monitor certain parts of the Messaging Server. For more information on monitoring the Messaging Server refer to Monitoring Messaging Server.

This information describes how to enable SNMP support for the Messaging Server. It also gives an overview of the type of information provided by SNMP. Note that it does not describe how to view this information from an SNMP client. Refer to your SNMP client documentation for details on how to use it to view SNMP-based information.

This information also describes some of the data available from the Messaging Server SNMP implementation, but complete MIB details are available from RFC 2788 (http://www.faqs.org/rfcs/rfc2788.html) and RFC 2789 (http://www.faqs.org/rfcs/rfc2788.html).

Topics:

SNMP Implementation

Messaging Server implements two standardized MIBs, the Network Services Monitoring MIB (RFC 2788) and the Mail Monitoring MIB (RFC 2789). The Network Services Monitoring MIB provides for the monitoring of network services such as POP, IMAP, HTTP, and SMTP servers. The Mail Monitoring MIB provides for the monitoring of MTAs. The Mail Monitoring MIB allows for monitoring both the active and historical state of each MTA channel. The active information focuses on currently queued messages and open network connections (for example, counts of queued messages, source IP addresses of open network connections), while the historical information provides cumulative totals (for example, total messages processed, total inbound connections).

> **Note**
> For a complete listing of Messaging Server SNMP monitoring information, refer to RFC 2788 and RFC 2789.

SNMP is supported on platforms running Oracle Solaris and Red Hat Linux. Messaging Server on the Oracle Solaris 9 Operating System uses Solstice Enterprise Agents (SEA). Starting with the Oracle Solaris 10 Operating System, Messaging Server supports the open source Net-SNMP monitoring framework, relegating the Oracle Solaris 9 Solstice Enterprise Agents (SEA) technology to legacy (end of support life) status. Additionally, Net-SNMP is widely used on Linux platforms. Messaging Server will use its Net-SNMP-based SNMP subagent on Oracle Solaris 10 and later as well as Linux platforms.

With the adoption of the Net-SNMP framework, Messaging Server's SNMP subagent provides new functionality:

- Support for SNMP versions 2c and 3. This support is provided by the Net-SNMP framework. The former SNMP technology, Solstice Enterprise Agents, only provided support for SNMP version 1.

Enhanced security features and access controls are the primary benefit of these two versions of SNMP.

- The subagent may be configured to run as a "standalone" SNMP agent. This provides sites with additional means of isolating their various SNMP agents running on the same system.
- Multiple "instances" of Messaging Server running on the same system may concurrently be monitored. This support is provided through either Item 2 above, or through the use of SNMP version 3 "context names". This allows for SNMP monitoring of Messaging Server in failover clusters.

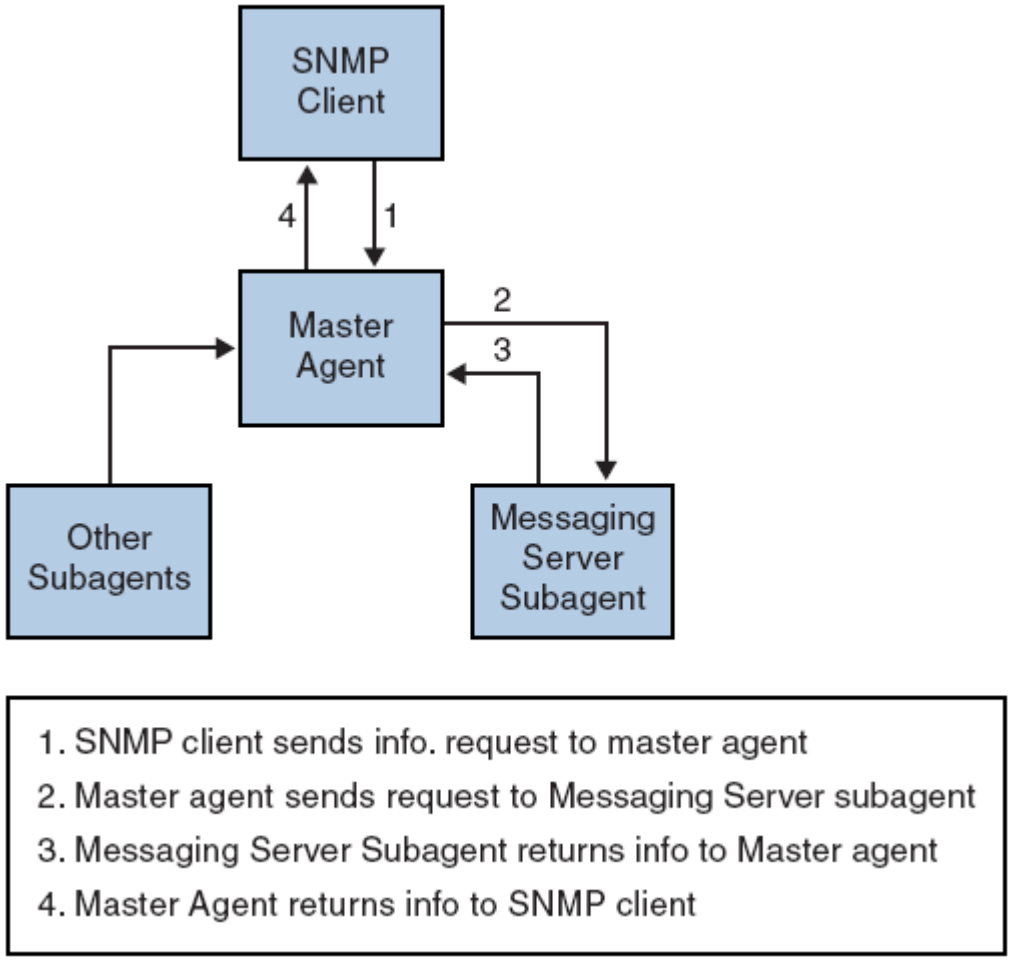
Limitations of the Messaging Server SNMP support are as follows:

- Only one instance of Messaging Server per host computer can be monitored via SNMP on Oracle Solaris 9.
- The SNMP support is for monitoring only. No SNMP management is supported.
- No SNMP traps are implemented. (RFC 2788 provides similar functionality without using traps.)

SNMP Operation in the Messaging Server

The Messaging Server SNMP process is an SNMP subagent which, upon startup, registers itself with the platform's native SNMP master agent. SNMP requests from clients go to the master agent. The master agent then forwards any requests destined for the Messaging Server to the Messaging Server subagent process. The Messaging Server subagent process then processes the request and relays the response back to the client via the master agent. This process is shown in the following figure.

SNMP Information Flow



Configuring SNMP Support for Oracle Solaris 10

> **Note**
> SNMP is not supported on Solaris 11. For a workaround, contact support.

By default, SNMP monitoring is disabled within Messaging Server. This default is chosen in an attempt to minimize the number of services presented by a default Messaging Server configuration. Do not interpret this default as meaning that there is a performance penalty incurred by using SNMP monitoring. Indeed, Messaging Server's SNMP support consumes very little resources and is intended to have minimal impact upon messaging server. The upshot of all of this is, of course, that one time configuration steps are required before using Messaging Server's SNMP support. Additionally, the default configuration of the platform's Net-SNMP master agent, `snmpd`, typically needs to be changed in order to run subagents such as Messaging Servers. This change is the topic of the next section.

Net-SNMP Configuration

Messaging Server's Net-SNMP based SNMP subagent uses the AgentX protocol to communicate with the platform's SNMP master agent (RFC 2741). The Net-SNMP master agent, `snmpd`, must be configured to permit the use of the AgentX protocol. To do this, ensure that the platform's `snmpd.conf` file contains the following line:

```
master agentx
```

If that line is not present, then add it and then restart the `snmpd` daemon. Note that sending a SIGHUP signal to the daemon is not sufficient. Once the `snmpd` daemon has been restarted, look for the UNIX domain socket which `snmpd` creates for AgentX communications. On Oracle Solaris and Linux Red Hat systems, this socket by default appears as the special file `/var/agentx/master`; however, its location and name may be changed via the `snmpd.con` file.

The Oracle Solaris 10 `snmpd` configuration is as follows:

```
%cp /etc/sma/snmp/snmpd.conf /etc/sma/snmp/snmpd.conf.save
% cat >> /etc/sma/snmp/snmpd.conf
# Messaging Server's subagent requires the AgentX protocol
master agentx
^D
% cat >> /etc/sma/snmp/snmpd.conf
% ls -al /var/agentx/
srwxrwxrwx 1 root root 0 Aug 9 13:58 /var/agentx/master
```

Additionally, on Red Hat Enterprise Linux AS 3 systems, the default `snmpd.conf` file restricts the information which may be viewed by the "public" SNMP community. It is therefore necessary to either remove that restriction or to extend it to include the MIBs served out by Messaging Server's subagent. For initial testing, doing the latter is recommended. This is accomplished by including the OID subtrees mib-2.27 and mib-2.28 in view named "systemview" as shown in below. For actual deployment, each site must take their overall security policy into consideration. Note that the information provided by the SNMP subagent is "read only".

```
% cp /etc/snmp/snmpd.conf /etc/snmp/snmpd.conf.save
% cat >>/etc/snmp/snmpd.conf
# Messaging Server's subagent requires the AgentX protocol
master agentx
# Messaging Server's subagent exports mib-2.27 and .28
# Add the mib-2.27 and .28 OID subtrees to the systemview
view systemview included .1.3.6.1.2.1.27
view systemview included .1.3.6.1.2.1.28
^D
% /sbin/service snmpd restart
% ls -al /var/agentx/master
srwxr-xr-x 1 root root 0 Aug 8 21:20 /var/agentx/master
```

If you will be using SNMP v3 context names to distinguish between the MIBs of different instances of Messaging Server concurrently running on the same host computer, then you will also need to configure at least one SNMP v3 username and password for use with your SNMP v3 queries.

Messaging Server Subagent Configuration

For basic operation of Messaging Server's SNMP subagent, you need only enable it and issue a one time manual start command. Henceforth, whenever Messaging Server is started or stopped, the subagent will likewise be started or stopped. The necessary commands to effect this configuration on both Oracle Solaris and Red Hat Linux are as follows:

```
% configutil -o local.snmp.enable -v 1
% start-msg snmp
```

Once running, you can test the subagent from the command line with the `snmpwalk` command. See the screen shots below an example appropriate to Oracle Solaris and Red Hat Linux. Note that the files `rfc2248.txt` and `rfc2249.txt` are copies of the Network Services and MTA MIBs. On Oracle Solaris systems, these files may also be found in the `/etc/sma/snmp/mibs/` directory under the names `NETWORK-SERVICES-MIB.txt` and `MTA-MIB.txt`. It is not necessary provide these files to the `snmpwalk` tool; however, doing so permits `snmpwalk` to print names for each of the MIB variables rather than their numeric object identifiers (OIDs).

Basic testing on Oracle Solaris:

```
% d=/opt/SUNWmsgsr/examples/mibs /usr/sfw/bin/snmpwalk -v 1 -c public \
    -m +$D/rfc2248.txt:$D/rfc2249.txt 127.0.0.1 mib-2.27

NETWORK-SERVICES-MIB::applName.1 = STRING: /opt/SUNWmsgsr MTA on
mail.siroe.com
...
% D=/opt/SUNWmsgsr/examples/mibs /usr/sfw/bin/snmpwalk -v 1 -c public \
     -m +$D/rfc2248.txt:$D/rfc2249.txt 127.0.0.1 mib-2.28

MTA-MIB::mtaReceivedMessages.1 = Counter32: 1452
MTA-MIB::mtaStoredMessages.1 = Gauge32: 21
...
```

Basic testing on Red Hat Linux:

```
% export d=/opt/sun/messaging/examples/mibs
% /usr/bin/snmpwalk -v 1 -c public \
     -m +$D/rfc2248.txt:$D/rfc2249.txt 127.0.0.1 mib-2.27
NETWORK-SERVICES-MIB::applName.1 = STRING: /opt/sun/messaging MTA on
mail.siroe.com
...
% /usr/bin/snmpwalk -v 1 -c public \
     -m +$D/rfc2248.txt:$D/rfc2249.txt 127.0.0.1 mib-2.28
MTA-MIB::mtaReceivedMessages.1 = Counter32: 21278
MTA-MIB::mtaStoredMessages.1 = Gauge32: 7
...
```

Running as a Standalone SNMP Agent

Before configuring Messaging Server's SNMP subagent to run as a standalone SNMP agent, you must first decide which Ethernet interface and UDP port you wish for it to listen on for SNMP requests. By default, it will listen on all available Ethernet interfaces using UDP port 161. In most cases, you will want to change the port number so as to not interfere with the platform's SNMP master agent, `snmpd`. In some circumstances such as HA failover you will also want to change the Ethernet interface from all available interfaces – `INADDR_ANY` – to a specific interface identified by its IP address. These two concepts, Ethernet interface and UDP port, are controlled via the `local.snmp.listenaddr` and `local.snmp.port` options.

Once choices for the Ethernet interface and UPD port have been made, the `local.snmp.standalone` option should have its value set to one and the subagent restarted. Once restarted, it will operate as a SNMP agent independent of `snmpd` and any subagents.

For example, to run as a standalone agent listening on UDP port 9161 of the Ethernet interface with IP address 10.53.1.37, issue the commands shown below.

Configuring to run as a standalone agent:

```
% configutil -o local.snmp.port -v 9161
% configutil -o local.snmp.listenaddr -v 10.53.1.37
% configutil -o local.snmp.standalone -v 1
% stop-msg snmp
% start-msg snmp
% snmpwalk -v 1 -c public 10.53.1.37:9161 .
SNMPv2-SMI::mib-2.27.1.1.2.1 = STRING: "/opt/SUNWmsgsr MTA on mail.siroe.com"
...
```

Monitoring Multiple Instances of Messaging Server

Two techniques for monitoring multiple instances of Messaging Server running on the same host computer are herein discussed. The first technique, running the subagent in standalone mode, is well suited to high-availability failover (HA) configurations in which the individual instances of Messaging Server may dynamically move between host computers. The second technique, the use of SNMP v3 context names, has some limited benefit in situations where multiple instances of Messaging Server are confined to a single system and it is desirable to limit the number of IP addresses polled by SNMP monitoring software (for example, when licensing of the monitoring software has a per IP address cost component). This latter technique may also be used in HA failover settings but would require polling just as many IP addresses as the standalone mode technique.

Using Standalone Agents for High-availability Failover

In a high-availability failover setting where SNMP monitoring of Messaging Server is desired, it is recommended that you run Messaging Server's SNMP subagent as a standalone agent as described in Running as a Standalone SNMP Agent. When the subagents are run in standalone mode, each HA instance of Messaging Server should have its `local.snmp.listenaddr` option set to the value of that instance's failover IP address. To simplify management, each instance should use the same UDP port, but that port should be distinct from those used by the `snmpd` daemons running on each of the physical cluster hosts. Typically those daemons will be using UDP port 161 so explicitly specify a different port number with the `local.snmp.port` option.

When Messaging Server's SNMP support is configured as recommended here, a monitoring station can monitor each instance of Messaging Server via its failover IP address or hostname regardless of which physical cluster host the instance is running on. Moreover, you are assured that Messaging Server's standalone SNMP agents will not conflict with one another as each listens only on its own virtual Ethernet interface identified by that instance's unique failover IP address. (These virtual Ethernet interfaces are automatically created by the HA failover framework.) Owing to the careful selection of a UDP port, the agents do not conflict with the `snmpd` daemons running on systems within the cluster.

Distinguishing Multiple Instances Through SNMP v3 Context Names

While there is no downside to using Messaging Server's SNMP support in standalone mode as described in Running as a Standalone SNMP Agent, it is recognized that some sites may prefer to use a more traditional subagent mode while still maintaining the capability of monitoring multiple instances of Messaging Server running concurrently on the same system. For instance, an SNMP monitoring system whose licensing model limits the number of IP addresses which may be polled. To achieve this goal, continue to run Messaging Server's SNMP subagent with `local.snmp.standalone` set to zero. Additionally, configure each instance of Messaging Server to use a distinct SNMP v3 context name by specifying a non-zero value for the `local.snmp.enablecontextname` option. If a context name different than the value of `service.defaultdomain` is desired, then set the desired name with the `local.snmp.contextname` option. Once each instance of Messaging Server's SNMP subagent is restarted, they can then be monitored via SNMP v3 queries which include the proper context names. The MIBs of two instances of Messaging Server running on the same system are distinguished by the instance's SNMP v3 context name and so no MIB object identifier (OID) conflicts will arise.

Messaging Server's Net-SNMP-based SNMP Subagent Options

The following options apply only to Messaging Server's Net-SNMP based SNMP subagent. That subagent is used on Oracle Solaris platforms running Oracle Solaris 10 and later as well as Linux platforms. The options described below do not apply to the legacy SNMP subagent supplied for Oracle Solaris platforms running Oracle Solaris 9 and earlier operating systems.

The options described below are `configutil` options. As such, their values are inspected with a command of the form:

```
% configutil -o <option-name>
```

where *option-name* is the name of the option to display the value of. To set or change an option's value, use a command of the form

```
% configutil -o <option-name> -v <option-value>
```

where <*option-value*> is the value to be set. Changes to these options require a restart to take effect:

```
% stop-msg snmp
% start-msg snmp
```

What follows is a description of each option along with its default value.

SNMP Subagent Options

Option (Default)	Description
local.snmp.enable (0)	The Messaging Server SNMP subagent will only run when this option is given a value of `1` or `true` in which case Messaging Server will automatically stop and start the subagent as part of its normal startup and shutdown procedures. By default this option is set to zero which disables operation of the subagent. Before enabling the subagent, ensure that the platform's master agent has been properly configured as described in Running as a Standalone SNMP Agent.
local.snmp.standalone (0)	Messaging Server's SNMP support normally runs as a SNMP subagent, receiving SNMP requests via the platform's SNMP master agent, `snmpd`. This operational mode is the default and is selected by giving this option a value of 0 or `false`. However, as described in Running as a Standalone SNMP Agent, the subagent may run in a "standalone" mode whereby it operates as a SNMP agent independent of `snmpd`. When run in standalone mode, the subagent, now an SNMP agent, listens directly for SNMP requests on the Ethernet interface and UDP port specified by, respectively, the `local.snmp.listenaddr` and `local.snmp.port` options. To run in this standalone mode, specify a value of 1 or TRUE for this option. Running in standalone mode does not interfere with other SNMP master or subagents running on the system.
local.snmp.listenaddr (INADDR_ANY)	Hostname or IP address of the Ethernet interface to listen for SNMP requests on when running in standalone mode. By default, all available interfaces are listened on. This corresponds to specifying the value `INADDR_ANY`. A specific interface may be selected by specifying either the IP address or hostname associated with that interface. The interface may be either a physical interface or a virtual interface. This option is ignored when `local.snmp.standalone` is set to 0 or FALSE.

local.snmp.cachettl (30)	Time to live (TTL) in seconds for cached monitoring data. This option controls how long the subagent will report the same monitoring data before refreshing that data with new information obtained from Messaging Server. With the exception of message loop information, data is cached for no longer than 30 seconds by default. Loop information, as determined by scanning for `.HELD` files, is updated only once every 10 minutes. That because of the resource cost of scanning all the on-disk message queues. Note that the subagent does not continually update its monitoring data: it is only updated upon receipt of an SNMP request and the cached data has expired (that is, outlived its TTL). If the TTL is set to 30 seconds and SNMP requests are made only every five minutes, then each SNMP request will cause the subagent to obtain fresh data from Messaging Server. That is, data from Messaging Server will be obtained only once every five minutes. If, on the other hand, SNMP requests are made every 10 seconds, then the subagent will respond to some of those requests with cached data as old as 29 seconds; Messaging Server will be polled only once every 30 seconds.
local.snmp.servertimeout (5)	The subagent determines the operational status of each monitored service by actually opening TCP connections to each service and undergoing a protocol exchange. This timeout value, measured in seconds, controls how long the subagent will wait for a response to each step in the protocol exchange. By default, a timeout value of five seconds is used.
local.snmp.directoryscan (1)	Use this option to control whether or not the subagent performs scans of the on-disk message queues for `.HELD` message files and the oldest message files. That information corresponds to the `mtaGroupLoopsDetected`, `mtaGroupOldestMessageStored` and `mtaGroupOldestMessageId` MIB variables. When this option has the value 1 or `true`, then a cache of this information is maintained and updated as needed. Sites with thousands of queued messages, that are not interested in these particular MIB variables should consider setting this option's value to 0 or `false`.
local.snmp.enablecontextname (0)	The subagent has the ability to register its MIBs under an SNMP v3 *context name*. When this is done, the MIBs may only be requested by a SNMP v3 client which specifies the context name in its SNMP request. Use of context names allows multiple, independent subagents to register Network Services and MTA MIBs under the same OID tree (that is, under the same SNMP master agent). See Monitoring Multiple Instances of Messaging Server for further information. To enable the use of SNMP v3 context names, specify a value of 1 or `true` for this option. When that is done, the subagent will default to using the value of the `service.defaultdomain` option for its context name. To use a different value for the context name, use the `local.snmp.contextname` option.
local.snmp.contextname (service.defaultdomain)	When the use of SNMP v3 context names has been enabled with `local.snmp.enablecontextname`, this option may be used to explicitly set the context name used by the subagent for its MIBs. The values supplied for this option are string values and must be appropriate for use as a SNMP v3 context name. This option is ignored when `local.snmp.enablecontextname` has the value 0 or `false`.

Monitoring from an SNMP Client

The base OIDs for RFC 2788 (http://www.faqs.org/rfcs/rfc2788.html) and RFC 2789 (http://www.faqs.org/rfcs/rfc2788.html) are

mib-2.27 = 1.3.6.1.2.1.27

mib-2.28 = 1.3.6.1.2.1.28

Point your SNMP client at those two OIDs and access as the “public” SNMP community.

If you wish to load copies of the MIBs into your SNMP client, ASCII copies of the MIBs are located in the *msg-svr-base*`/lib/config-templates` directory under the file names `rfc2788.mib` and `rfc2789.mib`. For directions on loading those MIBs into your SNMP client software, consult the SNMP client software documentation. The `SnmpAdminString` data type used in those MIBs may not be recognized by some older SNMP clients. In that case, use the equivalent files `rfc2248.mib` and `rfc2249.mib` also found in the same directory.

SNMP Information from the Messaging Server

This section summarizes the Messaging Server information provided via SNMP. It consists of the following subsection:

- applTable
- assocTable
- mtaTable
- mtaGroupTable
- mtaGroupAssociationTable
- mtaGroupErrorTable

For detailed information refer to the individual MIB tables in RFC 2788 (http://www.faqs.org/rfcs/rfc2788.html) and RFC 2789 (http://www.faqs.org/rfcs/rfc2788.html). Note that the RFC/MIB terminology refers to the messaging services (MTA, HTTP, and so on) as *applications* (`appl`), Messaging Server network connections as *associations* (`assoc`), and MTA channels as *MTA groups* (`mtaGroups`).

Note that on platforms where more than one instance of Messaging Server may be concurrently monitored, there may then be multiple sets of MTAs and servers in the `applTable`, and multiple MTAs in the other tables.

> **Note**
> The cumulative values reported in the MIBs (for example, total messages delivered, total IMAP connections, and so on) are reset to zero after a reboot.

Each site will have different thresholds and significant monitoring values. A good SNMP client will allow you to do trend analysis and then send alerts when sudden deviations from historical trends occur.

applTable

The `applTable` provides server information. It is a one-dimensional table with one row for the MTA and an additional row for each of the following servers, if enabled: WebMail HTTP, IMAP, POP, SMTP, and SMTP Submit. This table provides version information, uptime, current operational status (up, down, congested), number of current connections, total accumulated connections, and other related data.

Below is an example of data from `applTable` (mib-2.27.1.1).

```
applTable:

applName.1  = mailsrv-1  MTA on mailsrv-1.west.sesta.com      (1)
applVersion.1 = 5.1
applUptime.1 = 7322                          (2)
applOperStatus.1 = up                        (3)
applLastChange.1 = 7422                      (2)
applInboundAssociations.1 =                  (5)
applOutboundAssociations.1 =                 (2)
applAccumulatedInboundAssociations.1 = 873
applAccumulatedOutboundAssociations.1 = 234
applLastInboundActivity.1 = 1054822           (2)
applLastOutboundActivity.1 = 1054222          (2)
applRejectedInboundAssociations.1 = 0         (4)
applFailedOutboundAssociations.1 = 17
applDescription.1 = Sun Java System Messaging Server 6.1
applName.2 1 = mailsrv-1 HTTP WebMail svr. mailsrv-1.sesta.com (1)
...
applName.3 = mailsrv-1 IMAP server on mailsrv-1.west.sesta.com
...
applName.4 = mailsrv-1 POP server on mailsrv-1.west.sesta.com
...
applName.5 = mailsrv-1 SMTP server on mailsrv-1.west.sesta.com
...
applName.6 = mailsrv-1 SMTP Submit server on mailsrv-1.west.sesta.com
...
```

Notes:

1. The application (`.appl*`) suffixes (`.1`, `.2`, and so on) are the row numbers, `applIndex`. `applIndex` has the value 1 for the MTA, value 2 for the HTTP server, and so on. Thus, in this example, the first row of the table provides data on the MTA, the second on the POP server, and so on.
   The name after the equal sign is the name of the Messaging Server instance being monitored. In this example, the instance name is mailsrv-1.
2. These are SNMP TimeStamp values and are the value of `sysUpTime` at the time of the event. `sysUpTime`, in turn, is the count of hundredths of seconds since the SNMP master agent was started.
3. The operational status of the HTTP, IMAP, POP, SMTP, and SMTP Submit servers is determined by actually connecting to them via their configured TCP ports and performing a simple operation using the appropriate protocol (for example, a HEAD request and response for HTTP, a `HELO` command and response for SMTP, and so on). From this connection attempt, the status—`up` (1), `down` (2), or `congested` (4)—of each server is determined.
   Note that these probes appear as normal inbound connections to the servers and contribute to the value of the `applAccumulatedInboundAssociations` MIB variable for each server.
   For the MTA, the operational status is taken to be that of the Job Controller. If the MTA is shown to be up, then the Job Controller is up. If the MTA is shown to be down, then the Job Controller is down. This MTA operational status is independent of the status of the MTA's Service Dispatcher. The operational status for the MTA only takes on the value of up or down. Although the Job Controller does have a concept of "congested," it is not indicated in the MTA status.
4. For the HTTP, IMAP, and POP servers the `applRejectedInboundAssociations` MIB variable indicates the number of failed login attempts and not the number of rejected inbound connection attempts.

applTable Usage

Monitoring server status (`applOperStatus`) for each of the listed applications is key to monitoring each server.

If it's been a long time since the MTA last inbound activity as indicated by `applLastInboundActivity`, then something may be broken preventing connections. If `applOperStatus=2` (down), then the monitored service is down. If `applOperStatus=1` (up), then the problem may be elsewhere.

assocTable

This table provides network connection information to the MTA. It is a two-dimensional table providing information about each active network connection. Connection information is not provided for other servers.

Below is an example of data from applTable (mib-2.27.2.1).

```
assocTable:

    assocRemoteApplication.1.1  = 129.146.198.167        (1)
    assocApplicationProtocol.1.1 = applTCPProtoID.25     (2)
    assocApplicationType.1.1 = peerinitiator(3)          (3)
    assocDuration.1.1 = 400                              (4)
...
```

Notes:

In the `.x.y` suffix (1.1), `x` is the application index, `applIndex`, and indicates which application in the `applTable` is being reported on. In this case, the MTA. The `y` serves to enumerate each of the connections for the application being reported on.

1. The source IP address of the remote SMTP client.
2. This is an OID indicating the protocol being used over the network connection. `aplTCPProtoID` indicates the TCP protocol. The `.n` suffix indicates the TCP port in use and `.25` indicates SMTP which is the protocol spoken over TCP port 25.
3. It is not possible to know if the remote SMTP client is a user agent (UA) or another MTA. As such, the subagent always reports `peer-initiator`; `ua-initiator` is never reported.
4. This is an SNMP `TimeInterval` and has units of hundredths of seconds. In this example, the connection has been open for 4 seconds.

assocTable Usage

This table is used to diagnose active problems. For example, if you suddenly have 200,000 inbound connections, this table can let you know where they are coming from.

mtaTable

This is a one-dimensional table with one row for each MTA in the `applTable`. Each row gives totals across all channels (referred to as groups) in that MTA for select variables from the `mtaGroupTable`.

Below is an example of data from `applTable` (mib-2.28.1.1).

```
mtaTable:

    mtaReceivedMessages.1 = 172778
    mtaStoredMessages.1 = 19
    mtaTransmittedMessages.1 = 172815
    mtaReceivedVolume.1 = 3817744
    mtaStoredVolume.1 = 34
    mtaTransmittedVolume.1 = 3791155
    mtaReceivedRecipients.1 = 190055
    mtaStoredRecipients.1 = 21
    mtaTransmittedRecipients.1 = 3791134
    mtaSuccessfulConvertedMessages.1 = 0 (1)
    mtaFailedConvertedMessages.1 = 0
    mtaLoopsDetected.1 = 0               (2)
```

Notes:

The `.x` suffix (.1) provides the row number for this application in the `applTable`. In this example, `.1` indicates this data is for the first application in the `applTable`. Thus, this is data on the MTA.

1. Only takes on non-zero values for the conversion channel.
2. Counts the number of `.HELD` message files currently stored in the MTA's message queues.

mtaTable Usage

If mtaLoopsDetected is not zero, then there is a looping mail problem. Locate and diagnose the .HELD files in the MTA queue to resolve the problem.

If the system does virus scanning with a conversion channel and rejects infected messages, then mtaSuccessfulConvertedMessages will give a count of infected messages in addition to other conversion failures.

mtaGroupTable

This two-dimensional table provides channel information for each MTA in the `applTable`. This information includes such data as counts of stored (that is, queued) and delivered mail messages. Monitoring the count of stored messages, `mtaGroupStoredMessages`, for each channel is critical: when the value becomes abnormally large, mail is backing up in your queues.

Below is an example of data from `mtaGroupTable` (mib-2.28.2.1).

```
mtaGroupTable:

mtaGroupName.1.1 = tcp_intranet                1
        ...
mtaGroupName.1.2 = ims-ms
        ...
mtaGroupName.1.3 = tcp_local
    mtaGroupDescription.1.3 = mailsrv-1 MTA tcp_local channel
    mtaGroupReceivedMessages.1.3 = 12154
    mtaGroupRejectedMessages.1.3 = 0
    mtaGroupStoredMessages.1.3 = 2
    mtaGroupTransmittedMessages.1.3 = 12148
    mtaGroupReceivedVolume.1.3 = 622135
    mtaGroupStoredVolume.1.3 = 7
    mtaGroupTransmittedVolume.1.3 = 619853
    mtaGroupReceivedRecipients.1.3 = 33087
    mtaGroupStoredRecipients.1.3 = 2
    mtaGroupTransmittedRecipients.1.3 = 32817
    mtaGroupOldestMessageStored.1.3 = 1103
    mtaGroupInboundAssociations.1.3 = 5
    mtaGroupOutboundAssociations.1.3 = 2
    mtaGroupAccumulatedInboundAssociations.1.3 = 150262
    mtaGroupAccumulatedOutboundAssociations.1.3 = 10970
    mtaGroupLastInboundActivity.1.3 = 1054822
    mtaGroupLastOutboundActivity.1.3 = 1054222
    mtaGroupRejectedInboundAssociations.1.3 = 0
    mtaGroupFailedOutboundAssociations.1.3 = 0
    mtaGroupInboundRejectionReason.1.3 =
    mtaGroupOutboundConnectFailureReason.1.3 =
    mtaGroupScheduledRetry.1.3 = 0
    mtaGroupMailProtocol.1.3 = applTCPProtoID.25
    mtaGroupSuccessfulConvertedMessages.1.3 = 03     2
    mtaGroupFailedConvertedMessages.1.3 = 0
    mtaGroupCreationTime.1.3 = 0
    mtaGroupHierarchy.1.3 = 0
    mtaGroupOldestMessageId.1.3 = <01IFBV8AT8HYB4T6UA@red.iplanet.com>
    mtaGroupLoopsDetected.1.3 = 0                    3
    mtaGroupLastOutboundAssociationAttempt.1.3 = 1054222
```

Notes:

In the `.x.y` suffix (example: `1.1, 1.2. 1.3`), `x` is the application index, `applIndex`, and indicates which application in the `applTable` is being reported on. In this case, the MTA. The `y` serves to enumerate each of the channels in the MTA. This enumeration index, `mtaGroupIndex`, is also used in the `mtaGroupAssociationTable` and `mtaGroupErrorTable` tables.

1. The name of the channel being reported on. In this case, the tcp_intranet channel.
2. Only takes on non-zero values for the conversion channel.
3. Counts the number of `.HELD` message files currently stored in this channel's message queue.

mtaGroupTable Usage

Trend analysis on **Rejected** and **Failed** might be useful in determining potential channel problems.

A sudden jump in the ratio of mtaGroupStoredVolume to mtaGroupStoredMessages could mean that a large junk mail is bouncing around the queues.

A large jump in mtaGroupStoredMessages could indicate unsolicited bulk email is being sent or that delivery is failing for some reason.

If the value of mtaGroupOldestMessageStored is greater than the value used for the undeliverable message notification times (notices channel keyword) this may indicate a message which cannot be processed even by bounce processing. Note that bounces are done nightly so you will want to use mtaGroupOldestMessageStored > (maximum age + 24 hours) as the test.

If mtaGroupLoopsDetected is greater than 0, a mail loop has been detected.

mtaGroupAssociationTable

This is a three-dimensional table whose entries are indices into the `assocTable`. For each MTA in the `applTable`, there is a two-dimensional sub-table. This two-dimensional sub-table has a row for each channel in the corresponding MTA. For each channel, there is an entry for each active network connection which that channel has currently underway. The value of the entry is the index into the `assocTable` (as indexed by the entry's value and the `applIndex` index of the MTA being looked at). This indicated entry in the `assocTable` is a network connection held by the channel.

In simple terms, the `mtaGroupAssociationTable` table correlates the network connections shown in the `assocTable` with the responsible channels in the `mtaGroupTable`.

Below is an example of data from `mtaGroupAssociationTable` (mib-2.28.3.1).

```
mtaGroupAssociationTable:

    mtaGroupAssociationIndex.1.3.1 = 1 1
    mtaGroupAssociationIndex.1.3.2 = 2
    mtaGroupAssociationIndex.1.3.3 = 3
    mtaGroupAssociationIndex.1.3.4 = 4
    mtaGroupAssociationIndex.1.3.5 = 5
    mtaGroupAssociationIndex.1.3.6 = 6
    mtaGroupAssociationIndex.1.3.7 = 7
```

Notes:

In the `.x.y.z` suffix, `x` is the application index, `applIndex`, and indicates which application in the `applTable` is being reported on. In this case, the MTA. The `y` indicates which channel of the `mtaGroupTable` is being reported on. In this example, 3 indicates the tcp_local channel. The `z` serves to enumerate the associations open to or from the channel.

1. The value here is an index into the `assocTable`. Specifically, `x` and this value become, respectively, the values of the `applIndex` and `assocIndex` indices into the `assocTable`. Or, put differently, this is saying that (ignoring the `applIndex`) the first row of the `assocTable` describes a network connection controlled by the tcp_local channel.

mtaGroupErrorTable

This is another three-dimensional table which gives the counts of temporary and permanent errors encountered by each channel of each MTA while attempting delivery of messages. Entries with index values of 4000000 are temporary errors while those with indices of 5000000 are permanent errors. Temporary errors result in the message being re-queued for later delivery attempts; permanent errors result in either the message being rejected or otherwise returned as undeliverable.

Below is an example of data from `mtaGroupErrorTable` (mib-2.28.5.1).

```
mtaGroupErrorTable:

    mtaGroupInboundErrorCount.1.1.4000000 1 = 0
    mtaGroupInboundErrorCount.1.1.5000000 = 0
    mtaGroupInternalErrorCount.1.1.4000000 = 0
    mtaGroupInternalErrorCount.1.1.5000000 = 0
    mtaGroupOutboundErrorCount.1.1.4000000 = 0
    mtaGroupOutboundErrorCount.1.1.5000000 = 0
    mtaGroupInboundErrorCount.1.2.4000000 1 = 0
    ...
    mtaGroupInboundErrorCount.1.3.4000000 1 = 0
    ...
```

Notes:

1. In the `.x.y.z` suffix, `x` is the application index, `applIndex`, and indicates which application in the `applTable` is being reported on. In this case, the MTA. The `y` indicates which channel of the `mtaGroupTable` is being reported on. In this example, 1 specifies the tcp_intranet channel, 2 the ims-ms channel, and 3 the tcp_local channel. Finally, the `z` is either 4000000 or 5000000 and indicates, respectively, counts of temporary and permanent errors encountered while attempting message deliveries for that channel.

mtaGroupErrorTable Usage

A large jump in error count may likely indicate an abnormal delivery problem. For instance, a large jump for a tcp_ channel may indicate a DNS or network problem. A large jump for the ims_ms channel may indicate a delivery problem to the message store (for example, a partition is full, `stored` problem, and so on).

Chapter 33. Administering Event Notification Service in Messaging Server

Administering Event Notification Service in Oracle Communications Messaging Server

This information describes what you need to do to enable the Event Notification Service Publisher (ENS Publisher) and administer Event Notification Service (ENS) in Messaging Server.

Topics:

For more information on ENS and ENS APIs, see *Unified Communications Suite Event Notification Service Guide*.

ENS Publisher in Messaging Server

The Event Notification Service (ENS) is the underlying publish-and-subscribe service. ENS acts as a dispatcher used by Communications Suite applications as a central point of collection for certain types of *events* that are of interest to them. Events are changes to the value of one or more properties of a resource. Any application that wants to know when these types of events occur registers with ENS, which identifies events in order and matches notifications with subscriptions. ENS and iBiff (the ENS publisher for Messaging Server) are bundled with Messaging Server.

Loading the ENS Publisher on Messaging Server 7 Update 4

Beginning with **Messaging Server 7 Update 4**, significant updates have been made to IMAP IDLE, which affect the ENS configuration:

- IMAP IDLE is enabled by default when using the Event Notification Service (ENS).
- ENS is enabled by default (`local.ens.enable` set to 1).
- No configuration is required to load the ENS publisher because the `ms-internal` instance is automatically loaded and configured. Therefore, there is no need to create a separate `ms-internal` instance.
- If you want to configure parameters for the pre-loaded `ms-internal` default instance (as described in Administering Event Notification Service), set them with the `ms-internal` instance name. For example, `local.store.notifyplugin.ms-internal.`*settings*.
- IMAP IDLE now only works using ENS, because the ability to use IMAP IDLE with JMQ has been removed.
- For existing installations, customers should, at a minimum, ensure that the ENS server is enabled by setting the `configutil` parameter `local.ens.enable` to 1 and restarting the Messaging Server:

```
configutil -o local.ens.enable -v 1
```

Loading the ENS Publisher on Messaging Server 7 Update 3 and Prior Releases

In Messaging Server 7 Update 3 and prior releases, by default ENS is enabled, however, iBIFF is not loaded.

To subscribe to notifications in Messaging Server, you need to load the `libibiff` file on the Messaging Server host then stop and restart the messaging server.

Perform the following steps from the command line. In these steps, the location of the Messaging Server installation directory is *msg-svr-base*, and the Messaging Server user is *inetuser*. Typical values for these variables are `/opt/SUNWmsgsr`, and `mailsrv`, respectively.

1. As `mailsrv`, run the `configutil` utility to load the `libibiff` file.

```
cd <msg-svr-base>
./configutil -o "local.store.notifyplugin" -v
"<msg-svr-base>/lib/libibiff"
```

2. As `root`, stop then restart the messaging server.
   `cd` *msg-svr-base*`/sbin`
   `./stop-msg`
   `./start-msg`
3. You are now ready to receive notifications through ENS. See Running Sample Event Notification Service Programs in Messaging Server 7 Update 3.

Running Sample Event Notification Service Programs in Messaging Server 7 Update 3

Messaging Server 7 Update 3 and prior releases contain sample programs to help you learn how to receive notifications. These sample programs are located in the *msg-svr-base*`/examples` directory.

> **Note**
> Starting with Messaging Server 7 Update 4, ENS is redesigned to be simpler and more efficient such that it is enabled by default. The new sample programs in Messaging Server 7 Update 4, (`ens_pub.c` and `ens_sub.c`), and API (`ens.h`) can be obtained by contacting Oracle Support.

To Run the Sample ENS Programs in Messaging Server 7 Update 3

1. Change to the *msg-svr-base*`/examples` directory.
2. Using a C compiler, compile the `apub` and `asub` examples using the `Makefile.sample` file. Set your library search path to include the *msg-svr-base*`/examples` directory.
3. Once the programs have been compiled, you can run them as follows in separate windows:
   `apub localhost 7997`
   `asub localhost 7997`
   Whatever is typed in the `apub` window should appear on the `asub` window. Also, if you use the default settings, all iBiff notifications should appear in the `asub` window.
4. To receive notifications published by iBiff, write a program similar to `asub.c`
   For more information on the sample programs, and writing your own programs for ENS, see *Unified Communications Suite Event Notification Service Guide*.

Note
Once you set your library search path to include the *msg-svr-base*`/lib` directory, you can no longer stop and start the directory server. The workaround is to remove the entry from the library search path.

Administering Event Notification Service

Administering ENS consists of starting and stopping the service, and changing the configuration parameters to control the behavior of the iBiff publisher for ENS.

Starting and Stopping ENS

If desired, you can use the `start-msg ens` and `stop-message ens` commands to start and stop the ENS server.

Event Notification Service Configuration Parameters

The `local.store.notifyplugin.*` configuration parameters control the behavior of iBiff. Use the `configutil` utility program to set these parameters. For a list of parameters, see http://msg.wikidoc.info/index.php/Configutil_Reference#local.store.notifyplugin.

The following information is specific to Messaging Server 7 and later versions:

To enable ENS, set `local.ens.enable` to `1`, for example, `configutil -o local.ens.enable -v 1`, and restart Messaging Server.

The following information is specific to Messaging Server 7 Update 3 and earlier versions:

If `service.listenaddr` is set to a specific value, the ENS host also needs to be set with `local.store.notifyplugin.enshost`.

The following information is specific to Messaging Server 7 Update 4 and later versions:

The following `configutil` parameters are now marked `OBSOLETE` and are no longer valid:

```
service.imap.ensidle
service.imap.idle
service.imap.idle.jmq.library
service.imap.idle.jmqhost
service.imap.idle.jmqpassword
service.imap.idle.jmqport
service.imap.idle.jmqtopic
service.imap.idle.jmquserid
local.store.notifyplugin.enshost
local.store.notifyplugin.ensport
local.store.notifyplugin.enseventkey
local.store.notifyplugin.noneinbox.enable
```

For the `service.imap.*` parameters, you may remove these obsolete parameters from their configuration, although the only impact is to maintain a clean configuration.

Options can be removed by using the `-d` switch with `configutil`. For example:

```
configutil -o service.imap.ensidle -d
```

Customers who do not use ENS or JMQ for external notification systems can also delete `local.store.notifyplugin`, because an appropriate default value is now configured.

If `service.listenaddr` is set to a specific value, the ENS host also needs to be set with `local.store.notifyplugin.ms-internal.enshost`.

If you had set any of the `local.store.notifyplugin.*` parameters to a non-default value, you have to set them again under the new name, `notifyplugin.`*`eventtarget.option`* syntax, when you upgrade to Messaging Server 7 Update 4.

Chapter 34. Short Message Service (SMS)

Short Message Service (SMS)

This information describes how to implement the Short Message Service (SMS) on the Oracle Communications Messaging Server.

Topics:

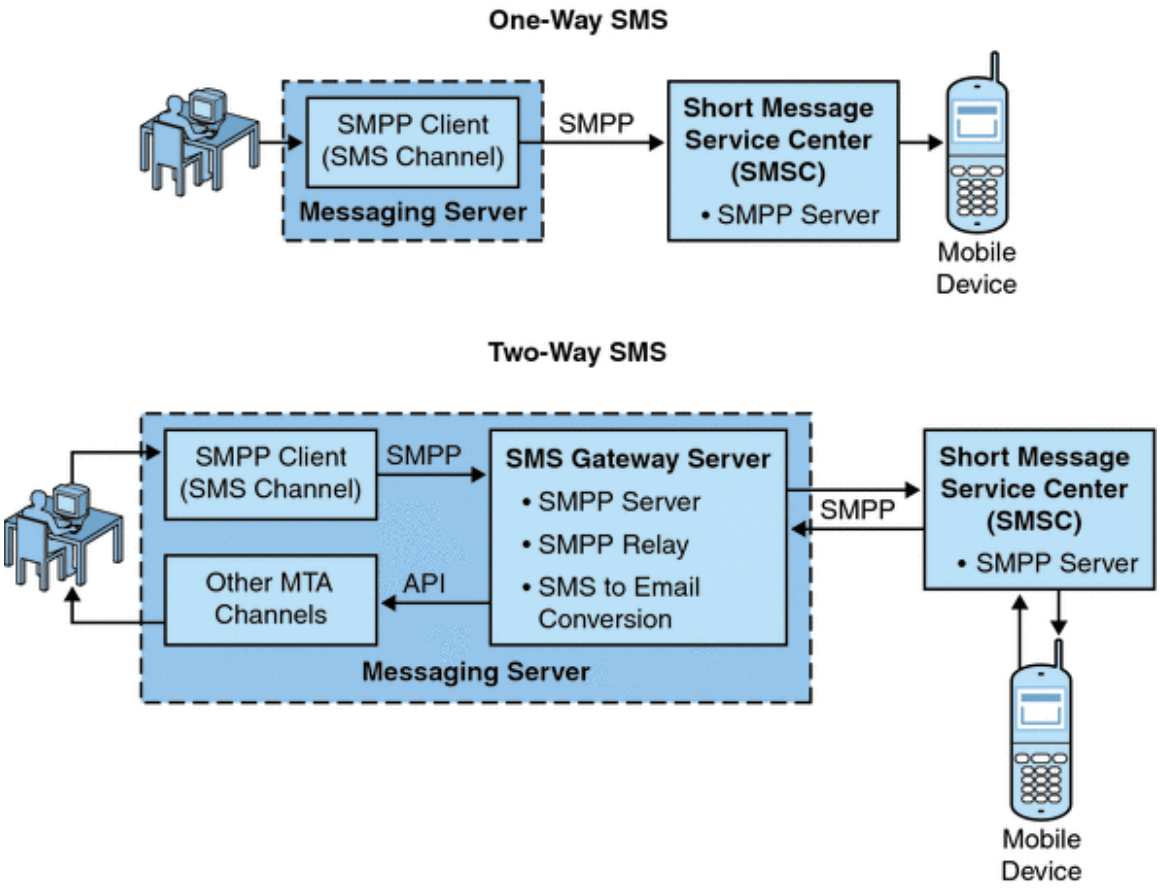
C.1 Introduction

Messaging Server implements email-to-mobile and mobile-to-email messaging using a Short Message Service (SMS). SMS can be configured to be either one-way (email-to-mobile only) or two-way (both email-to-mobile and mobile-to-email). To enable one-way service only, you must add and configure the SMS channel. To enable two-way service, you must add and configure the SMS channel, and in addition, configure the SMS Gateway Server.

For both one- and two-way SMS, the generated SMS messages are submitted to a Short Message Service Center (SMSC) using the Short Message Peer to Peer (SMPP) protocol. Specifically, the SMSC must provide a V3.4 or later SMPP server that supports TCP/IP.

Figure C-1 illustrates the logical flow of messages for both one-way and two-way SMS.

Figure C-1 Logical Flow For One-Way and Two-Way SMS



C.1.1 One-Way SMS

To enable one-way service, the Messaging Server implements an SMPP client (the MTA SMS channel) that communicates with remote SMSCs. The SMS channel converts enqueued email messages to SMS messages as described in C.2.2 The Email to SMS Conversion Process of multipart MIME messages as well as character set translation issues.

Operating in this capacity, the SMS channel functions as an (SMPP) External Short Message Entity (ESME).

C.1.1.1 Two-Way SMS

Two-Way SMS enables the mail server not only to send email to remote devices, but allows for receiving replies from the remote devices and for remote device email origination.

Enabling two-way SMS service requires both the MTA SMS channel (SMPP client), as explained in the previous topic, and the SMS Gateway Server. Messaging Server installs an SMS Gateway Server as part of its general installation process, which you must then configure. The SMS Gateway Server performs two functions:

- SMPP relay
  The SMS Gateway Server acts as a transparent SMPP client between the MTA SMS channel and SMSCs. However, in addition, while acting as a relay, the SMS Gateway Server generates unique SMS source addresses for relayed messages, and saves the message IDs returned by the remote SMSCs for later correlation with SMS notification messages.
- SMPP server
  The SMS Gateway Server acts as an SMPP server to receive mobile originated SMS messages, replies to prior email messages, and SMS notifications. The SMS Gateway Server extracts destination email addresses from the SMS messages using profiles that define the conversion

process. Profiles also describe how to handle notification messages returned by remote SMSCs in response to previously sent email-to-mobile messages.

> **Note -**
> Messaging Server does not support the two-way SMS on the Windows platform.

C.1.2 Requirements

This manual assumes that you have read Logica CMG's SMPP specification, and the SMPP documentation for your SMSC.

In order to implement SMS, you must have the following:

- Messaging Server 6 or greater. (One-way SMS is also implemented in iPlanet Messaging Server 5.2.)
- The SMSC must support SMPP V3.4, or later, over TCP/IP and there must be TCP/IP connectivity between the host running Messaging Server and the SMSC.

For storage planning information for the SMS Gateway Server, see C.6 SMS Gateway Server Storage Requirements

C.2 SMS Channel Theory of Operation

The SMS channel is a multi-threaded channel which converts queued email messages to SMS messages and then submits them for delivery to an SMSC.

The following channel operation topics are covered in this section:

C.2.1 Directing Email to the Channel

When the SMS channel is configured as per C.3 SMS Channel Configuration purposes of discussion, let us assume that the host name `sms.siroe.com` is a host name associated with the channel. In that case, email is directed to the channel with an address of the form:

`local-part``@sms.siroe.com`

in which *`local-part`* is either the SMS destination address (for example, a wireless phone number, pager ID, etc.) or an attribute-value pair list in the format:

```
/attribute1=value1/attribute2=value2/.../@sms.siroe.com
```

The recognized attribute names and their usages are given in Table C-1. These attributes allow for per-recipient control over some channel options.

Table C-1 SMS Attributes

Attribute Name	Attribute Value and Usage
ID	SMS destination address (for example, wireless phone number, pager ID, etc.) to direct the SMS message to. This attribute and associated value must be present.
FROM	SMS source address. Ignored when option USE_HEADER_FROM=0.
FROM_NPI	Use the specified NPI value. Ignored when option USE_HEADER_FROM=0.
FROM_TON	Use the specified TON value. Ignored when option USE_HEADER_FROM=0.
MAXLEN	The maximum, total bytes (that is, eight bit bytes) to place into the generated SMS message or messages for this recipient. The lower value of either MAXLEN and the value specified by the MAX_MESSAGE_SIZE channel option is used.
MAXPAGES	The maximum number of SMS messages to split the email message into for this recipient. The lower value of either MAXPAGES and the value specified by the MAX_PAGES_PER_MESSAGE channel option is used.
NPI	Specify a Numeric Plan Indicator (NPI) value for the destination SMS address specified with the ID attribute. See the description of the DEFAULT_DESTINATION_NPI channel option for information on the accepted values for this attribute. When this attribute is used, its value overrides the value given by the DEFAULT_DESTINATION_NPI channel option.
PAGELEN	Maximum number of bytes to place into a single SMS message for this recipient. The minimum of this value and that specified with the MAX_PAGE_SIZE channel option is used.
TO	Synonym for ID.
TO_NPI	Synonym for NPI.
TO_TON	Synonym for TON.
TON	Specify a Type of Number (TON) value for the destination SMS address given with the ID attribute. See the description of the DEFAULT_DESTINATION_TON channel option for information on the accepted values for this attribute. When this attribute is used, its value overrides the value given by the DEFAULT_DESTINATION_TON channel option.

Some example addresses:

```
123456@sms.siroe.com
/id=123456/@sms.siroe.com
/id=123456/maxlen=100/@sms.siroe.com
/id=123456/maxpages=1/@sms.siroe.com
```

For information on performing translations, validity checks, and other operations on the SMS destination address portion of the email address, see C.2.4 Site-defined Address Validity Checks and Translations

C.2.2 The Email to SMS Conversion Process

In order for email to be sent to a remote site, email must be converted to SMS messages that can be understood by the remote SMSCs. This section describes the process of converting an email message queued to the SMS channel to one or more SMS messages. As described below, options allow control over the maximum number of SMS messages generated, the maximum total length of those SMS messages, and the maximum size of any single SMS message. Only text parts (that is, MIME text content types) from the email message are used and the maximum number of parts converted may also be controlled.

Character sets used in the email message's header lines and text parts are all converted to Unicode and then converted to an appropriate SMS character set.

When there is no SMS_TEXT mapping table (see C.2.5 Site-defined Text Conversions) an email message queued to the SMS channel receives the processing illustrated in Figure C-2.

Figure C-2 SMS Channel Email Processing

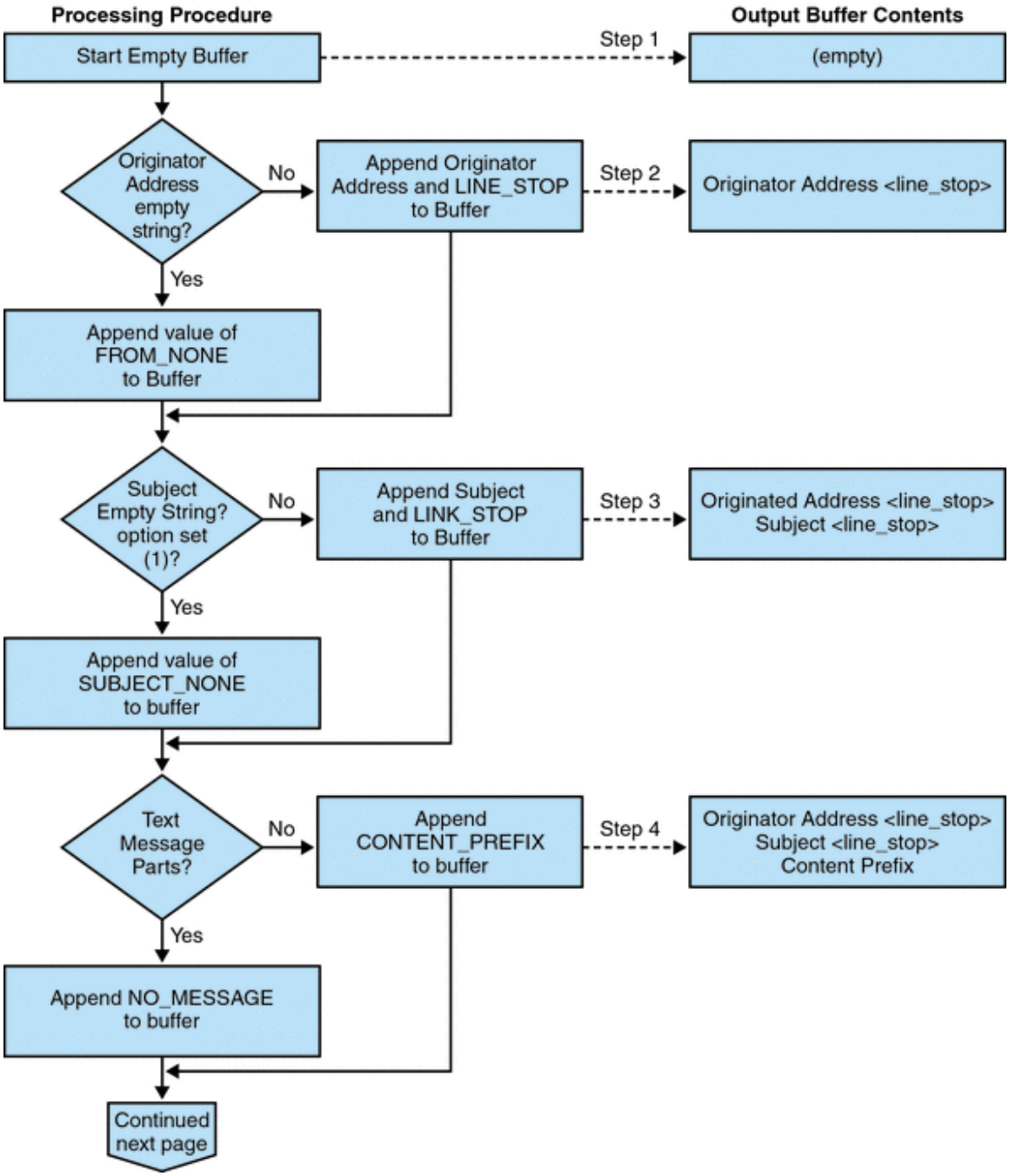
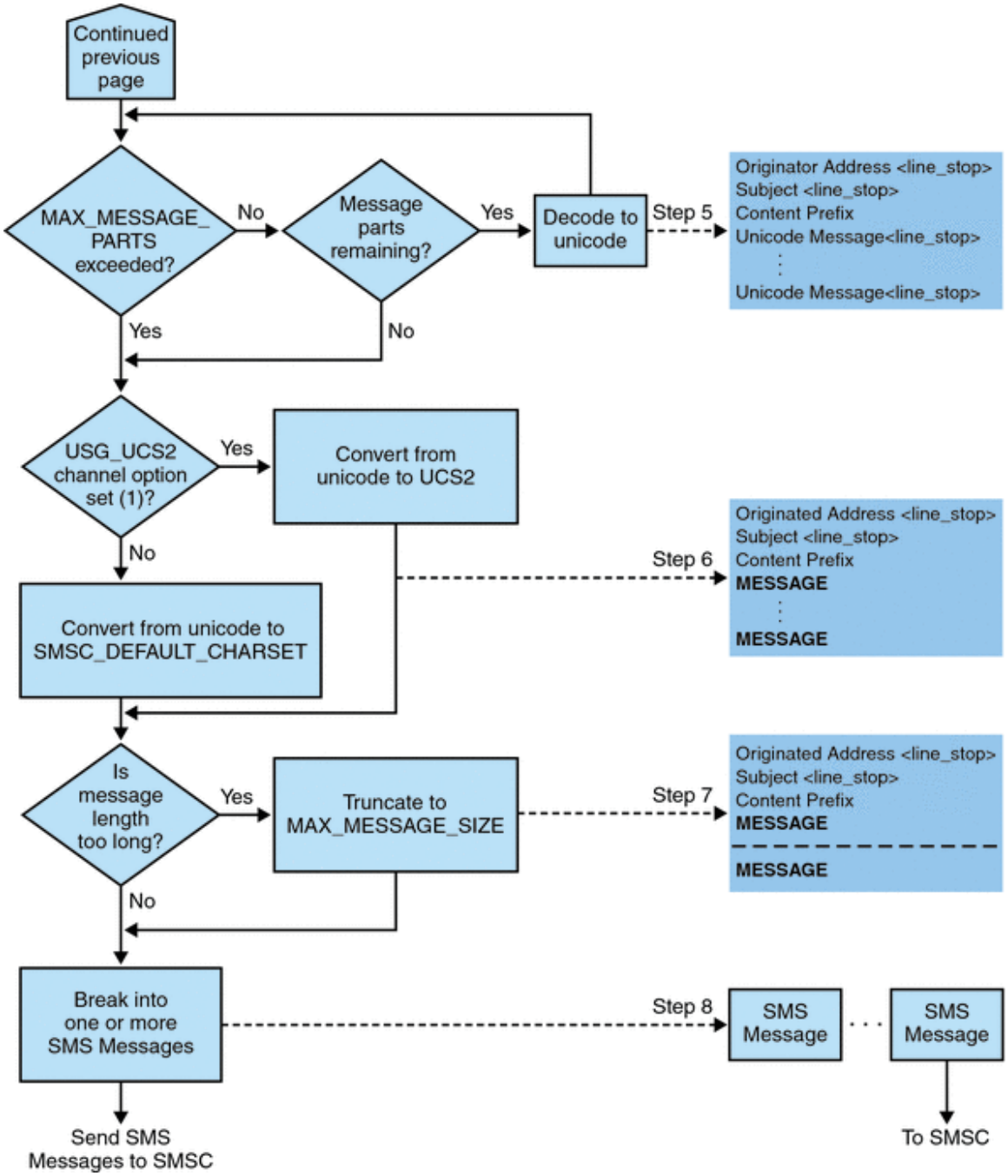


Figure C-3 SMS Channel Email Processing (*continued*)



The following steps correspond to the numbered boxes in Figure C-2:

1. An empty output buffer is started. The character set used for the buffer is Unicode.
2. The email message's originator address is taken from one of the following five sources, shown in decreasing order of preference:

```
1. Resent-from:
2. From:
3. Resent-sender:
4. Sender:
5. Envelope From:
```

   If the originator address is an empty string, then the value of the FROM_NONE channel option is instead appended to the buffer.

If, however, the originator address is a non-empty string, then the result of processing the FROM_FORMAT channel option, and the value of the LINE_STOP channel option are appended to the output buffer.

3. If a Subject: header line is not present or is empty, then the value of the SUBJECT_NONE option is appended to the output buffer.
   Otherwise, the result of processing the SUBJECT_FORMAT option, and the value of the LINE_STOP channel option are appended to the output buffer.
4. If there are no text message parts, then the value of the NO_MESSAGE channel option is appended to the output buffer.
   If there are text message parts, then the value of the CONTENT_PREFIX channel option is appended to the output buffer.
   Non-text message parts are discarded.
5. For each text part, while the MAX_MESSAGE_PARTS limit has not been reached, the text part is decoded to Unicode and appended to the buffer, along with the value of the LINE_STOP channel option.
6. The resulting output buffer is then converted from Unicode to either the SMSC's default character set or UCS2 (UTF-16). The SMSC's default character set is specified with the SMSC_DEFAULT_CHARSET option.
7. After being converted, it is then truncated to not exceed MAX_MESSAGE_SIZE bytes.
8. The converted string from C.2.2 The Email to SMS Conversion Process is then broken into one or more SMS messages, no single SMS message longer than MAX_PAGE_SIZE bytes. At most, MAX_PAGES_PER_MESSAGE SMS messages will be generated.

> **Note -**
> As an email message may have multiple recipients, Step 6 through Step 8 may need to be done for each recipient address which makes use of the MAXLEN, MAXPAGES, or PAGELEN attributes described in "Directing Email to the Channel," on page 4.

C.2.2.1 Sample Email Message Processing

For example, with the channel's default settings, the email message:

```
From: John Doe
To: 1234567@sms.siroe.com
Subject: Today&rsquo;s meeting
Date: Fri, 26 March 2001 08:17

The staff meeting is at 14:30 today in the big conference room.
```

Would be converted to the SMS message:

```
jdoe@siroe.com (Today's meeting) The staff meeting is at 14:30 today in the
big conference room.
```

A different set of option settings, that follows:

```
CONTENT_PREFIX=Msg:
FROM_FORMAT=From:${pa}
SUBJECT_FORMAT=Subj:$s
```

would instead produce:

```
From:John Doe Subj:Today's meeting Msg:The staff meeting is at 14:30 today in
```

```
the big conference room.
```

C.2.3 The SMS Message Submission Process

Once an email message has been converted to one or more SMS messages, with possibly different sets for each recipient, the SMS messages are then submitted to the destination SMSC. The submissions are effected using SMPP V3.4 over TCP/IP. The hostname (SMPP_SERVER) of the SMPP server is taken to be the official host name associated with the SMS channel; the TCP port (SMPP_PORT) to use is specified with the port channel keyword.

When there are messages to process, the channel is started. The channel binds to the SMPP server as a transmitter, presenting the credentials specified with the ESME_ channel options described in C.3.3.4 SMPP Options. Table C-2 lists the fields set in a BIND_TRANSMITTER PDU (Protocol Data Unit), and gives their values:

Table C-2 Fields in Generated in a BIND_TRANSMITTER PDU

Field	Channel Option	Value
system_id	ESME_SYSTEM_ID	default value is an empty string
password	ESME_PASSWORD	default value is an empty string
system_type	ESME_SYSTEM_TYPE	default value is an empty string
interface_version	n/a	0x34 indicating SMPP V3.4
addr_ton	ESME_ADDRESS_TON	default value is 0x00 indicating an unknown TON
addr_npi	ESME_ADDRESS_NPI	default value is 0x00 indicating an unknown NPI
addr_range	ESME_IP_ADDRESS	default value is an empty string

Note that the channel is multithreaded. Depending on how much mail there is to send, the channel may have multiple dequeue thread running. (There can even be multiple channel processes running.) Each thread does a BIND_TRANSMITTER and then on that TCP/IP connection, sends all of the SMS messages it has to send, and then sends an UNBIND, and then closes the connection. No attempt is made to hold a connection open for a period of idle time for potential reuse. If the remote SMPP server sends back a throttle error, then an UNBIND is issued, the TCP/IP connection is closed, and a new connection and BIND established. It behaves similarly if the remote SMPP server sends an UNBIND before it is finished sending its SMS messages.

The SMS messages are then submitted using SMPP SUBMIT_SM PDUs. If a permanent error is returned (for example, ESME_RINVDSTADR), then the email message is returned as undeliverable. If a temporary error is returned, then the email message is re-enqueued for a later delivery attempt. To clarify, a permanent error is one for which the condition is likely to exist indefinitely and for which repeated delivery attempts will have no positive effect, such as invalid SMS destination addresses. Whereas, a temporary error is one for which the condition is likely to not exist in the near future, such as a server down or server congested condition.

If the USE_HEADER_FROM option has the value 1, then the source address for the submitted SMS message is set. The value used is derived from the originating email message and is chosen to be the most likely (email) address to which any replies should be directed. Accordingly, the source address taken from one of the following seven sources, shown in decreasing order of preference:

```
1. Resent-reply-to:
2. Resent-from:
3. Reply-to:
4. From:
5. Resent-sender:
6. Sender:
7. Envelope From:
```

Note that the `Resent-reply-to:` and `Reply-to:` header lines are only considered if the USE_HEADER_REPLY_TO option has the value 1. The default value is the value 0. As such, only items 4, 6, and 7 are considered by the default configuration. Finally, since the source address in an SMS message is limited to 20 bytes, the source address chosen will be truncated if it exceeds that limit.

Table C-3 shows the mandatory fields set in a `SUBMIT_SM` PDU:

Table C-3 Mandatory Fields in Generated `SUBMIT_SM` PDUs

Field	Value
service_type	DEFAULT_SERVICE_TYPE channel option; default value is an empty string.
source_addr_ton	DEFAULT_SOURCE_TON channel option; if USE_HEADER_FROM=1, then this field is usually forced to the value 0x05 indicating an alphanumeric TON; otherwise, the default value is 0x01 indicating an international TON.
source_addr_npi	DEFAULT_SOURCE_NPI channel option; default value is 0x00.
source_addr	DEFAULT_SOURCE_ADDRESS channel option if USE_HEADER_FROM=0; otherwise, an alphanumeric string representing the originator of the email message.
dest_addr_ton	TON addressing attribute or DEFAULT_DESTINATION_TON channel option; default value is 0x01 indicating an international TON.
dest_addr_npi	NPI addressing attribute or DEFAULT_SOURCE_NPI channel option; default value is 0x00 indicating an unknown NPI.
dest_addr	Destination SMS address derived from the local part of the email envelope To: address; see C.2.1 Directing Email to the Channel.
esm_class	For one-way SMS, set to 0x03, indicating store and forward mode, default SMSC message type, and do not set reply path. For a two-way MSM message, set to 0x83.
protocol_id	0x00; unused for CDMA and TDMA; for GSM, 0x00 indicates no Internet, but SME-to-SME protocol.
priority_flag	0x00 for GSM & CDMA and 0x01 for TDMA, all indicating normal priority; See the description of the DEFAULT_PRIORITY channel option.
schedule_delivery_time	Empty string indicating immediate delivery.
validity_period	DEFAULT_VALIDITY_PERIOD channel option; default value is an empty string indicating that the SMSC's default should be used.
registered_delivery	0x00 indicating no registered delivery.
replace_if_present_flag	0x00 indicating that any previous SMS messages should not be replaced.
data_coding	0x00 for the SMSC's default character set; 0x08 for the UCS2 character set.
sm_default_msg_id	0x00 indicating not to use a pre-defined message.
sm_length	Length and content of the SMS message; see C.2.2 The Email to SMS Conversion Process
short_message	Length and content of the SMS message; see C.2.2 The Email to SMS Conversion Process

Table C-4 shows the optional fields in a SUBMIT_SM PDU:

Table C-4 Optional Fields in Generated SUBMIT_SM PDUs

Field	Value
privacy	See the description of the DEFAULT_PRIVACY channel keyword; default is to not provide this field unless the email message has a Sensitivity: header line
sar_refnum	See the description of the USE_SAR channel keyword; default is to not provide these fields
sar_total	See sar_refnum above.
sar_seqnum	See sar_refnum above.

The channel remains bound to the SMPP server until either it has no more SMS messages to submit (the message queue is empty), or MAX_PAGES_PER_BIND has been exceeded. In the latter case, a new connection is made and bind operation performed if there remain further SMS messages to send.

Note that the SMS channel is multithreaded. Each processing thread in the channel maintains its own TCP connection with the SMPP server. For example, if there are three processing threads each with SMS messages to submit, then the channel will have three open TCP connections to the SMPP server. Each connection will bind to the SMPP server as a transmitter. Moreover, any given processing thread will only have one outstanding SMS submission at a time. That is, a given thread will submit an SMS message, then wait for the submission response (that is, SUBMIT_SM_RESP PDU) before submitting another SMS message.

C.2.4 Site-defined Address Validity Checks and Translations

Sites may wish to apply validity checks or translations to SMS destination addresses encoded in the recipient email addresses described in C.2.1 Directing Email to the Channel

- Strip non-numeric characters (for example, translating 800.555.1212 to 8005551212)
- Prepend a prefix (for example, translating 8005551212 to +18005551212)
- Validate for correctness (for example, 123 is too short)

The first two tasks can be done specifically with the DESTINATION_ADDRESS_NUMERIC and DESTINATION_ADDRESS_PREFIX channel options. In general, all three of these tasks, and others can be implemented using mapping tables: either mapping table callouts in the rewrite rules or by means of a FORWARD mapping table. Using a mapping table callout in the rewrite rules will afford the most flexibility, including the ability to reject the address with a site-defined error response. The remainder of this section will focus on just such an approach – using a mapping table callout from the rewrite rules.

Let us suppose that destination addresses need to be numeric only, be 10 or 11 digits long, and be prefixed with the string "+1". This can be accomplished with the following rewrite rule

```
sms.siroe.com      ${X-REWRITE-SMS-ADDRESS,$U}@sms.siroe.com
sms.siroe.com       $?Invalid SMS address
```

The first rewrite rule above calls out to the site-define mapping table named X-REWRITE-SMS-ADDRESS. That mapping table is passed the local part of the email address for inspection. If the mapping process decides that the local part is acceptable, then the address is accepted and rewritten to the SMS channel. If the mapping process does not accept the local part, then the next rewrite rule is applied. Since it is a $? rewrite rule, the address is rejected with the error text "Invalid SMS address".

The X-REWRITE-SMS-ADDRESS mapping table is shown below. It performs the necessary validation steps for local parts in either attribute-value pair list format or just a raw SMS destination address.

```
X-VALIDATE-SMS-ADDRESS

! Iteratively strip any non-numeric characters
   $_*$[$ -/:-~]%*  $0$2$R
! Accept the address if it is of the form 1nnnnnnnnnn or nnnnnnnnnn
! In accepting it, ensure that we output +1nnnnnnnnnn
   1%%%%%%%%%%      +1$0$1$2$3$4$5$6$7$8$9$Y
   %%%%%%%%%%       +1$0$1$2$3$4$5$6$7$8$9$Y
! We didn&rsquo;t accept it and consequently it&rsquo;s invalid
   *                 $N

X-REWRITE-SMS-ADDRESS
    */id=$_*/*       $C$0/id=$|X-VALIDATE-SMS-ADDRESS;$1|/$2$Y$E
    */id=$_*/*       $N
    *                $C$|X-VALIDATE-SMS-ADDRESS;$0|$Y$E
    *                $N
```

With the above set up, be sure that DESTINATION_ADDRESS_NUMERIC option has the value 0 (the default). Otherwise, the "+" will be stripped from the SMS destination address.

C.2.5 Site-defined Text Conversions

Sites may customize Steps 1 - 6 described in C.2.2 The Email to SMS Conversion Process a mapping table in the MTA's mapping file.

The name of the mapping table should be *SMS_Channel_TEXT* where *SMS_Channel*

is the name of the SMS channel; for example, `SMS_TEXT` if the channel is named `sms` or `SMS_MWAY_TEXT` if the channel is named `sms_mway`.

Two types of entries may be made in this mapping table. However, before explaining the format of those entries, let it be made clear that an understanding of how to use the mapping file is essential in order to understand how to construct and use these entries. An example mapping table is given after the description of these two types of entries.

Now, the two types of entries are:

- C.2.5.1 Message Header Entries
- C.2.5.2 Message Body Entries

C.2.5.1 Message Header Entries

These entries specify which message header lines should be included in an SMS message and how they should be abbreviated or otherwise converted. Only if a header line is successfully mapped to a string of non-zero length by one of these entries will it be included in the SMS message being generated. Each entry has the format

`H|`*pattern replacement-text*

If a message header line matches the pattern then it will be replaced with the replacement text replacement-text using the mapping file's pattern matching and string substitution facilities. The final result of mapping the header line will then be included in the SMS message provided that the metacharacter $Y was specified in the replacement text. If a header line does not match any pattern string, if it maps to a string of length zero, or if the $Y metacharacter is not specified in the replacement text, then the header line will be omitted from the SMS message. The two entries

```
H|From:* F:$0$Y
H|Subject:* S:$0$Y
```

cause the `From:` and `Subject:` header lines to be included in SMS messages with `From:` and `Subject:` abbreviated as `F:` and `S:`. The entries:

```
H|Date:* H|D:$0$R$Y
H|D:*,*%19%%*:*:* H|D:$0$ $5:$6$R$Y
```

cause the `Date:` header line to be accepted and mapped such that, for instance, the header line

```
Date: Wed, 16 Dec 1992 16:13:27 -0700 (PDT)
```

will be converted to

```
D: Wed 16:13
```

Very complicated, iterative mappings may be built. Sites wishing to set up custom filters will first need to understand how the mapping file works. The `H|` in the right-hand-side of the entry may be omitted, if desired. The `H|` is allowed in that side so as to cut down on the number of table entries required by sets of iterative mappings.

C.2.5.2 Message Body Entries

Body mappings are not supported.

C.2.5.3 Example SMS Mapping Table

An example `SMS_TEXT` mapping table is shown in Example C-1. The numbers inside parentheses at the end of each line correspond to the item numbers in the section titled Explanatory Text that follows this table.

Example C-1 Example SMS_TEXT Mapping Table.

```
SMS_TEXT

   H|From:*         H|F:$0$R$Y       (1)
   H|Subject:*      H|S:$0$R$Y       (1)
   H|F:*&lt;*>*     H|F:$1$R$Y       ()
   H|F:*(*)*        H|F:$0$2$R$Y     (2)
   H|F:*"*"*        H|F:$0$2$R$Y     (3)
   H|F:*@*          H|F:$0$R$Y       (4)
   H|%:$ *          H|$0:$1$R$Y      (5)
   H|%:*$           H|$0:$1$R$Y      (5)
   H|%:*$ $ *       H|$0:$1$ $2$R$Y  (6)
   B|*--*           B|$0-$1$R        (7)
   B|*..*           B|$0.$1$R        (7)
   B|*!!*           B|$0!$1$R        (7)
   B|*??*           B|$0?$1$R        (7)
   B|*$ $ *         B|$0$ $1$R       (6)
   B|$ *            B|$0$R           (5)
   B|*$             B|$0$R           (5)
```

Explanatory Text

The entries in the example SMS_TEXT mapping table above are explained below:

In the example above, the metacharacter $R is used to implement and control iterative application of the mappings. By iterating on these mappings, powerful filtering is achieved. For instance, the simple mappings to remove a single leading or trailing space (6) or reduce two spaces to a single space (7) become, when taken as a whole, a filter which strips all leading and trailing spaces and reduces all consecutive multiple spaces to a single space. Such filtering helps reduce the size of each SMS message.

1. These two entries cause `From:` and `Subject:` header lines to be included in an SMS message. `From:` and `Subject:` are abbreviated as, respectively, `F:` and `S:`. Some of the other entries may have further effects on `From:` and `Subject:` header lines.
   This entry will reduce a `From:` header line containing a <...> pattern to only the text within the angle brackets. For example:
   `F: "John C. Doe" <jdoe@siroe.com> (Hello)`
   will be replaced with:
   `F: jdoe@siroe.com`
2. This entry will remove, inclusively, everything inside of a (...) pattern in a `From:` header line. For example:
   `F: "John C. Doe" <jdoe@siroe.com> (Hello)`
   will be replaced with:
   `F: "John C. Doe" <jdoe@siroe.com>`
3. This entry will remove, inclusively, everything inside of a "..." pattern in a `From:` header line. For example:
   `F: "John C. Doe" <jdoe@siroe.com> (Hello)`
   will be replaced with:
   `F: <jdoe@siroe.com> (Hello)`
4. This entry will remove, inclusively, everything to the right of an at-sign, @, in a `From:` header line. For example:
   `F: "John C. Doe" <jdoe@siroe.com> (Hello)`
   will be replaced with:
   `F: "John C. Doe" <jdoe@`
5. These four entries remove leading and trailing spaces from lines in the message header and body.

6. These two entries reduce two spaces to a single space in lines of the message header and body.
7. These four entries reduce double dashes, periods, exclamation and question marks to single occurrences of the matching character. Again, this helps save bytes in an SMS message.

The order of the entries is very important. For instance, with the given ordering, the body of the message `From:` header line:

```
From: "John C. Doe" (Hello)
```

will be reduced to:

```
jdoe
```

The steps taken to arrive at this are as follows:

1. We begin with the From: header line:
   `From: "John C. Doe" (Hello)`
   The pattern in the first mapping entry matches this and produces the result:
   `F: "John C. Doe" (Hello)`
   The $R metacharacter in the result string causes the result string to be remapped.
2. The mapping is applied to the result string of the last step. This produces:
   `F: jdoe@siroe.com`

The $R in the mapping causes the entire set of mappings to be re-applied to the result of this step.

3. Next, the mapping is applied producing:
   `F: jdoe`
   The $R in the mapping causes the entire set of mappings to be re-applied to the result of this step.

4. Next, the mapping is applied producing:
   `F:jdoe`
   The $R in the mapping causes the entire set of mappings to be re-applied to the result of this step.

5. Since no other entries match, the final result string:
   `F:jdoe`
   is incorporated into the SMS message.

> **Note -**
> The `imsimta` test-mapping utility may be used to test a mapping table. For instance,
>
> ```
> # imsimta test -mapping -noimage_file
> -mapping_file=test.txt
> Enter table name: SMS_TEXT
> Input string: H|From: "John C. Doe"  (Hello)
> Output string: H|F:jdoe
> Output flags: [0,1,2,89]
> Input string: ^D
> #
> ```
>
> For further details on the `imsimta test` utility, see http://docs.sun.com/doc/819-4429/acmjb?a=view.

C.3 SMS Channel Configuration

This section gives directions on how to set up the SMS channel for both one-way (email-to-mobile) and two-way (email-to-mobile and mobile-to-email) functionality. The SMS channel is set up the same for both one-way and two-way functionality, with the exceptions noted in the topic C.3.7 Configuring the SMS Channel for Two-Way SMS

This section includes the following topics:

C.3.1 Adding an SMS Channel

Two steps are required to add an SMS channel to a Messaging Server configuration:

1. C.3.1.1 Adding the Channel Definition and Rewrite Rules.
2. C.3.2 Creating an SMS Channel Option File.

While there are no channel options which must be set in all situations, it is likely that one or more of the

following options may need to be set: ESME_PASSWORD, ESME_SYSTEM_ID, MAX_PAGE_SIZE, DEFAULT_SOURCE_TON, and DEFAULT_DESTINATION_TON. And, as described, the SMPP server's hostname or IP address and TCP port must be set either through the channel definition in the `imta.cnf` file or the channel option file.

You may configure more than one SMS channel, giving different characteristics to different SMS channels. See C.3.4 Adding Additional SMS Channels for further information on the use of multiple SMS channels.

Note for the instructions that follow: if you change the `imta.cnf` file you must recompile. if you change just a channel option file, there is no need to recompile.

Note also that the time before a channel change takes effect can differ depending on what the change is. Many channel option changes take effect in all channels started since the change was made, which may seem almost instantaneous since the Job Controller is often starting new channels. Some of the changes don't take effect until you recompile and restart the SMTP server. These options are processed as a message is enqueued to the channel and not when the channel itself runs.

C.3.1.1 Adding the Channel Definition and Rewrite Rules

To add the channel definition and rewrite rules, do the following:

To Add Channel Definition and Rewrite Rules

1. Before adding an SMS channel to the MTA's configuration, you need to pick a name for the channel. The name of the channel may be either `sms` or `sms_x` where `x` is any case-insensitive string whose length is between one and thirty-six bytes. For example, `sms_mway`.
2. To add the channel definition, edit the `imta.cnf` file located in the `installation-directory /config/` directory. At the bottom of the file add a blank line followed by the two lines:

   ```
   _channel-name_ port _p_ threaddepth _t_ \ backoff "pt2m" "pt5m" "pt10m"
   "pt30m" notices 1
   _smpp-host-name_
   ```

   where *channel-name* is the name you chose for the channel, *p* is the TCP port the SMPP server listens on, *t* is the maximum simultaneous number of SMPP server connections per delivery process, and *smpp-host-name* is the host name of the system running the SMPP server.
   For example, you might specify a channel definition as follows:

   ```
   sms_mway port 55555 threaddepth 20 \
   backoff "pt2m" "pt5m" "pt10m" "pt30m" notices 1
   smpp.siroe.com
   ```

   For instructions on how to calculate `threaddepth`, see C.3.1.2 Controlling the Number of Simultaneous Connections
   See C.3.5 Adjusting the Frequency of Delivery Retries for a discussion of the `backoff` and `notices` channel keywords.
   If you wish to specify an IP address rather than a host name, for `smpp-host-name`, specify a domain literal. For example, if the IP address is 127.0.0.1, then specify [127.0.0.1] for `smpp-host-name`. Alternatively, consider using the SMPP_SERVER channel option.

   > **Note -**
   > Starting with Sun Java System Messaging Server 6.1, the use of the `master` channel keyword has been deprecated. It is ignored if present.

3. Once the channel definition has been added, go to the top half of the file and add a rewrite rule in

this format:
`smpp-host-name $u@smpp-host-name`
For example,
`smpp.siroe.com $u@smpp.siroe.com`

4. Save the `imta.cnf` file.
5. Recompile the configuration with the `imsimta cnbuild` command.
6. Restart the SMTP server with the `imsimta restart dispatcher` command.
7. With the above configuration, email messages are directed to the channel by addressing them to `id@smpp-host-name` (for example, `123456@smpp.siroe.com`). See C.2.2 The Email to SMS Conversion Process for further information on addressing.
8. Optionally, if you wish to hide the SMPP server's host name from users or associate other host names with the same channel, then add additional rewrite rules. For instance, to associate `host-name-1` and `host-name-2` with the channel, add the following to rewrite rules:

```
host-name-1 $U%host-name-1@smpp-host-name
host-name-2 $U%host-name-2@smpp-host-name
```

For example, if the SMPP server's host name is `smpp.siroe.com` but you want users to address email to `id@sms.sesta.com`, then add the rewrite rule:
`sms.sesta.com $U%sms.sesta.com@smpp.siroe.com`
Note that the SMPP_SERVER and SMPP_PORT channel options will override the channel's official host name and `port` channel keyword settings. When the `SMPP_PORT` option is used, it is not necessary to also use the `port` keyword. The advantage of using these two options is that they can be put into effect and subsequently changed without needing to recompile the configuration. An additional use of the `SMPP_SERVER` option is described in C.3.4 Adding Additional SMS Channels.

C.3.1.2 Controlling the Number of Simultaneous Connections

The `threaddepth` channel keyword controls the number of messages to assign to each delivery thread within a delivery process. To calculate the total number of concurrent connections allowed, multiply the values of the two following options: `SMPP_MAX_CONNECTIONS` and `job_limit` (`SMPP_MAX_CONNECTIONS` * job_limit). The SMPP_MAX_CONNECTIONS option controls the maximum number of delivery threads in a delivery process. And, the `job_limit` option, for the Job Controller processing pool in which the channel is run, controls the maximum number of simultaneous delivery processes.

To limit the total number of concurrent connections, you must adjust appropriately either or both of these options. For instance, if the remote SMPP server allows only a single connection, then both `SMPP_MAX_CONNECTIONS` and `job_limit` must be set to `1`. When adjusting the values, it's preferable to allow `job_limit` to exceed `1`.

C.3.2 Creating an SMS Channel Option File

In general, a channel option file contains site-specific parameters required for the operation of the channel. A channel option file is not required for SMS. If you determine that one is necessary for your installation, store it in a text file in the `installation-directory/config/` directory. As with other channel option files, the name of the file takes the form:

`channel_name_option`

For instance, if the channel is named `sms_mway` then the channel option file is:

`installation-directory/config/sms_mway_option`

Each option is placed on a single line in the file using the format:

```
option_name=option_value
```

For example,

```
PROFILE=GSM
SMSC_DEFAULT_CHARSET=iso-8859-1
USE_UCS2=1
```

For a list of available SMS channel options and a description of each, see C.3.3 Available Options

C.3.3 Available Options

The SMS channel contains a number of options which divide into six broad categories:

- *Email to SMS conversion*: Options which control the email to SMS conversion process.
- *SMS Gateway Server Option:* Gateway profile option.
- *SMS fields*: Options which control SMS-specific fields in generated SMS messages.
- *SMPP protocol*: Options associated with the use of the SMPP protocol over TCP/IP.
- *Localization*: Options which allow for localization of text fields inserted into SMS messages.
- *Miscellaneous*: Debug and logging options.

These options are summarized in the table below, and described more fully in the sections which follow.

Table C-5 SMS Channel Options

Email to SMS Conversion Options		
Option	Description	Default
GATEWAY_NOTIFICATIONS	Specify whether or not to convert email notification messages to SMS messages.	`0`
MAX_MESSAGE_PARTS	Max. number of message parts to extract from an email message	`2`
MAX_MESSAGE_SIZE	Maximum number of bytes to extract from an email message	`960`
MAX_PAGE_SIZE	Maximum number of bytes to put into a single SMS message	`160`
MAX_PAGES_PER_MESSAGE	Max. number of SMS messages to break an email message into	`6`
ROUTE_TO	Route SMS messages to the specified IP host name.	
SMSC_DEFAULT_CHARSET	The default character set used by the SMSC.	`US-ASCII`
USE_HEADER_FROM	Set the SMS source address	`0`
USE_HEADER_PRIORITY	Control the use of priority information from the email message's header	`1`
USE_HEADER_REPLY_TO	Control the use of `Reply-to:` header lines when generating SMS source addresses	`0`
USE_HEADER_SENSITIVITY	Control the use of privacy information from the email message's header	`1`

USE_UCS2	Use the UCS2 character set in SMS messages when applicable	1
SMS Gateway Server Option		
GATEWAY_PROFILE	Match the gateway profile name configured in the SMS Gateway Server's configuration file, `sms_gateway.cnf`	N/A
SMS Fields Options		
DEFAULT_DESTINATION_NPI	Default NPI for SMS destination addresses	0x00
DEFAULT_DESTINATION_TON	Default TON for SMS destination addresses	0x01
DEFAULT_PRIORITY	Default priority setting for SMS messages	0=GSM, CDMA1=TDMA
DEFAULT_PRIVACY	Default privacy value flag for SMS messages	-1
DEFAULT_SERVICE_TYPE	SMS application service associated with submitted SMS messages	N/A
DEFAULT_SOURCE_ADDRESS	Default SMS source address	0
DEFAULT_SOURCE_NPI	Default NPI for SMS source addresses	0x00
DEFAULT_SOURCE_TON	Default TON for SMS source addresses	0x01
DEFAULT_VALIDITY_PERIOD	Default validity period for SMS messages	N/A
DESTINATION_ADDRESS_NUMERIC	Reduce the destination SMS address to only the characters 0 - 9	0
DESTINATION_ADDRESS_PREFIX	Text string to prefix destination SMS addresses with	N/A
PROFILE	SMS profile to use	GSM
USE_SAR	Sequence multiple SMS messages using the SMS `sar_` fields	0
SMPP Protocol Options		
ESME_ADDRESS_NPI	ESME NPI to specify when binding to the SMPP server	0x00
ESME_ADDRESS_TON	ESME TON to specify when binding to the SMPP server	0x00
ESME_IP_ADDRESS	IP address of the host running Messaging Server	N/A
ESME_PASSWORD	Password to present when binding to the SMPP server	N/A
ESME_SYSTEM_ID	System identification to present to the SMSC when binding	N/A
ESME_SYSTEM_TYPE	System type to present to the SMSC when binding	N/A
MAX_PAGES_PER_BIND	Maximum number of SMS messages to submit during a single session with an SMPP server	1024

REVERSE_ORDER	Transmission sequence of multi-part SMS messages	`0`
SMPP_MAX_CONNECTIONS	Maximum number of simultaneous SMPP server connections	`20`
SMPP_PORT	For one-way SMS, TCP port the SMPP server listens on. For two-way SMS, same TCP port used for the `LISTEN_PORT` for the SMPP relay.	N/A
SMPP_SERVER	For one-way SMS, host name of the SMPP server to connect to.For two-way SMS, set to point to the host name or IP address of the SMS Gateway server. If using the SMPP relay's `LISTEN_INTERFACE_ADDRESS` option, then be sure to use the host name or IP address associated with the specified network interface address.	N/A
TIMEOUT	Timeout for completion of reads and writes with the SMPP server	`30`
Localization Options		
CONTENT_PREFIX	Text to introduce the content of the email message	`Msg:`
DSN_DELAYED_FORMAT	Formatting string for delivery delay notifications	an empty string
DSN_FAILED_FORMAT	Formatting string for delivery failure notifications	see description
DSN_RELAYED_FORMAT	Formatting string for relay notifications.	see description
DSN_SUCCESS_FORMAT	Formatting string to successful delivery notifications.	see description
FROM_FORMAT	Text to display indicating the originator of the email message	`$a`
FROM_NONE	Text to display when there is no originator	N/A
LANGUAGE	(i-default) Language group to select text fields from	`i-default`
LINE_STOP	Text to place at the end of each line extracted from the email message	space character
NO_MESSAGE	Text to indicate that the message had no content	`]no message]`
SUBJECT_FORMAT	Text to display indicating the subject of the email message	`$s`
SUBJECT_NONE	Text to display when there is no subject for the email message	N/A
Miscellaneous Options		
DEBUG	Enable verbose debug output	6

LISTEN_CONNECTION_MAX	Maximum number of concurrent, inbound TCP connections to allow across all SMPP relay and server instantiations.	10,000
LOG_PAGE_COUNT	Controls the value recorded in the `mail.log` file's message size field to be page count instead of blocks.	0

C.3.3.1 Email to SMS Conversion Options

The following options control the conversion of email messages to SMS messages. The value range for the options are in parenthesis. In general, a given email message may be converted into one or more SMS messages. See C.2.2 The Email to SMS Conversion Process

GATEWAY_NOTIFICATIONS

(*0 or 1*) Specifies whether or not to convert email notifications to SMS notifications. Email notification messages must conform to RFCs 1892, 1893, 1894. The default value is `0`.

When `GATEWAY_NOTIFICATIONS=0`, such notifications are discarded and are not converted to SMS notifications.

To enable the notifications to be converted to SMS notifications, set `GATEWAY_NOTIFICATIONS=1`. When the option set to `1`, the localization options (`DSN_*_FORMAT`) control which notification types (success, failure, delay, relayed) are converted into SMS messages and sent through the gateway. (If the notification type has a value of an empty string, then that type notification is not converted into SMS messages.)

MAX_MESSAGE_PARTS

(*integer*) When converting a multi-part email message to an SMS message, only the first `MAX_MESSAGE_PARTS` number of text parts will be converted. The remaining parts are discarded. By default, `MAX_MESSAGE_PARTS` is 2. To allow an unlimited number of message parts, specify a value of -1. When a value of 0 is specified, then no message content will be placed into the SMS message. This has the effect of using only header lines from the email message (for example, `Subject:`) to generate the SMS message.

Note that an email message containing both text and an attachment will typically consist of two parts. Note further that only plain text message parts are converted. All other MIME content types are discarded.

MAX_MESSAGE_SIZE

(*integer, >= 10*) With this option, an upper limit may be placed on the total number of bytes placed into the SMS messages generated from an email message. Specifically, a maximum of `MAX_MESSAGE_SIZE` bytes will be used for the one or more generated SMS messages. Any additional bytes are discarded.

By default, an upper limit of 960 bytes is imposed. This corresponds to `MAX_MESSAGE_SIZE`=960. To allow any number of bytes, specify a value of zero.

The count of bytes used is made after converting the email message from Unicode to either the SMSC's default character set or UCS2. This means, in the case of UCS2, that a `MAX_MESSAGE_SIZE` of 960 bytes will yield, at most, 480 characters since each UCS2 character is at least two bytes long.

Note that the `MAX_MESSAGE_SIZE` and MAX_PAGES_PER_MESSAGE options both serve the same purpose: to limit the overall size of the resulting SMS messages. Indeed, MAX_PAGE_SIZE=960 and

MAX_PAGE_SIZE=160 implies MAX_PAGES_PER_MESSAGE=6. So why are there two different options? So as to allow control of the overall size or number of pages without having to consider the maximal size of a single SMS message, MAX_PAGE_SIZE. While this may not be important in the channel option file, it is important when using the C.2.1 Directing Email to the Channel or C.2.1 Directing Email to the Channel addressing attributes described in C.2.1 Directing Email to the Channel.

Finally, note that the smaller of the two limits of MAX_MESSAGE_SIZE and MAX_PAGE_SIZE * MAX_PAGES_PER_MESSAGE is used.

MAX_PAGE_SIZE

(*integer, >= 10*) The maximum number of bytes to allow in a single SMS message is controlled with the MAX_PAGE_SIZE option. By default, a value of 160 bytes is used. This corresponds to MAX_PAGE_SIZE=160.

MAX_PAGES_PER_MESSAGE

(*integer, 1 - 255*) The maximum number of SMS messages to generate for a given email message is controlled with this option. In effect, this option truncates the email message, only converting to SMS messages that part of the email message which fits into MAX_PAGES_PER_MESSAGE SMS messages. See the description of the MAX_PAGE_SIZE option for further discussion.

By default, MAX_PAGES_PER_MESSAGE is set to the larger of 1 or MAX_MESSAGE_SIZE divided by MAX_PAGE_SIZE.

ROUTE_TO

(*string*, *IP host name*, *1-64 bytes*) All SMS messages targeted to the profile will be rerouted to the specified IP host name using an email address of the form:

```
SMS-destination-address@route-to
```

where SMS-destination-address is the SMS message's destination address and the route-to is the IP host name specified with this option. The entire content of the SMS message is sent as the content of the resulting email message. The PARSE_RE_* options are ignored.

> **Note -**
> Use of PARSE_RE_* and ROUTE_TO options are mutually exclusive. Use of both in the same gateway profile is a configuration error.

SMSC_DEFAULT_CHARSET

(*string*) With this option, the SMSC's default character set may be specified. Use the character set names given in the file

```
installation-directory/config/charsets.txt
```

When this option is not specified, then US-ASCII is assumed. Note that the mnemonic names used in charsets.txt are defined in charnames.txt in the same directory.

When processing an email message, the header lines and text message parts are first decoded and then converted to Unicode. Next, the data is then converted to either the SMSC's default character set or UCS2, depending on the value of the USE_UCS2 option and whether or not the SMS message contains at least one glyph not found in the default SMSC character set. Note that the UCS2 character set is a 16-bit encoding of Unicode and is often referred to as UTF-16.

USE_HEADER_FROM

(*integer, 0-2*) Set this option to allow the `From:` address to be passed to the SMSC. The value indicates where the `From:` address is taken from and what format it will have. Table C-6 shows the allowable values and their meaning.

Table C-6 USE_HEADER_FROM Values

Value	Description
`0`	SMS source address never set from the `From:` address. Use attribute-value pair found
`1`	SMS source address set to `from-local@from-domain`, where the From: address is: @ `from-route:from-local@from-domain`
`2`	SMS source address set to `from-local`, where the From: address is: @ `from-route:from-local@from-domain`

USE_HEADER_PRIORITY

(*0 or 1*) This option controls handling of RFC 822 `Priority:` header lines. By default, information from the `Priority:` header line is used to set the resulting SMS message's priority flag, overriding the default SMS priority specified with the DEFAULT_PRIORITY option. This case corresponds to `USE_HEADER_PRIORITY=1`. To disable use of the RFC 822 `Priority:` header line, specify `USE_HEADER_PRIORITY=0`.

See the description of the `DEFAULT_PRIORITY` option for further information on the handling the SMS priority flag.

USE_HEADER_REPLY_TO

(*0 or 1*) When `USE_HEADER_FROM =1`, this option controls whether or not a `Reply-to:` or `Resent-reply-to:` header line is considered for use as the SMS source address. By default, `Reply-to:` and `Resent-reply-to:` header lines are ignored. This corresponds to an option value of 0. To enable consideration of these header lines, use an option value of 1.

Note that RFC 2822 has deprecated the use of `Reply-to:` and `Resent-reply-to:` header lines.

USE_HEADER_SENSITIVITY

(*0 or 1*) The `USE_HEADER_SENSITIVITY` option controls handling of RFC 822 `Sensitivity:` header lines. By default, information from the `Sensitivity:` header line is used to set the resulting SMS message's privacy flag, overriding the default SMS privacy specified with the DEFAULT_PRIVACY option. This case, which is the default, corresponds to `USE_HEADER_SENSITIVITY=1`. To disable use of RFC 822 Sensitivity: header lines, specify `USE_HEADER_SENSITIVITY=0`.

See the description of the DEFAULT_PRIVACY option for further information on the handling the SMS privacy flag.

USE_UCS2

(*0 or 1*) When appropriate, the channel will use the UCS2 character set in the SMS messages it generates. This is the default behavior and corresponds to `USE_UCS2=1`. To disable the use of the UCS2 character set, specify `USE_UCS2=0`. See the description of the SMSC_DEFAULT_CHARSET option for further information on character set issues.

Table C-7 Valid Values for USE_UCS2

USE_UCS2 Value	Result
1 (default)	The SMSC default character set will be used whenever possible. When the originating email message contains glyphs not in the SMSC default character set, then the UCS2 character set will be used.
0	The SMSC default character set will always be used. Glyphs not available in that character set will be represented by mnemonics (for example, "AE" for AE-ligature).

C.3.3.2 SMS Gateway Server Option

GATEWAY_PROFILE

The name of the gateway profile in the SMS Gateway Server configuration file, `sms_gateway.cnf`.

C.3.3.3 SMS Options

The following options allow for specification of SMS fields in generated SMS messages.

DEFAULT_DESTINATION_NPI

(*integer, 0 - 255*) By default, destination addresses will be assigned an NPI (Numeric Plan Indicator) value of zero. With this option, an alternate integer value in the range 0 to 255 may be assigned. Typical NPI values include those found in Table C-8 that follows:

Table C-8 Numeric Plan Indicator Values

Value	Description
0	Unknown
1	ISDN (E.163, E.164)
3	Data (X.121)
4	Telex (F.69)
6	Land Mobile (E.212)
8	National
9	Private
10	ERMES
14	IP address (Internet)
18	WAP client ID
>= 19	Undefined

Values for this option may be specified in one of three ways:

- A decimal value (for example, 10).
- A hexadecimal value prefixed by "0x" (for example, 0x0a).
- One of the following case-insensitive text strings (the associated decimal value is shown in parentheses): data (3), default (0), e.163 (1), e.164 (1), e.212 (6), ermes (10), f.69 (4), Internet (14), ip (14), isdn (1), land-mobile (6), national (8), private (9), telex (4), unknown (0), wap (18), x.121 (3).

DEFAULT_DESTINATION_TON

(*integer, 0 - 255*) By default, destination addresses will be assigned a TON (Type of Number) designator value of zero. With this option, an alternate integer value in the range 0 to 255 may be assigned. Typical TON values include those found in Table C-9 that follows:

Table C-9 Typical TON Values

Value	Description
0	Unknown
1	International
2	National
3	Network specific
4	Subscriber number
5	Alphanumeric
6	Abbreviated
>=7	Undefined

Values for this option may be specified in one of three ways:

- A decimal value (for example, 10)
- A hexadecimal value prefixed by "0x" (for example, 0x0a)
- One of the following case-insensitive text strings (the associated decimal value is shown in parentheses): abbreviated (6), alphanumeric (5), default (0), international (1), national (2), network-specific (3), subscriber (4), unknown (0).

DEFAULT_PRIORITY

(integer, 0 - 255) SMS messages have a mandatory priority field. The interpretation of SMS priority values is shown in Table C-10 that follows:

Table C-10 SMS Priority Values Interpreted for Each SMS Profile Type

Value	GSM	TDMA	CDMA
0	Non-priority	Bulk	Normal
1	Priority	Normal	Interactive
2	Priority	Urgent	Urgent
3	Priority	Very urgent	Emergency

With this option, the default priority to assign to SMS messages may be specified. When not specified, a default priority of 0 is used for `PROFILE=GSM` and `CDMA`, and a priority of 1 for PROFILE=TDMA.

Note that if USE_HEADER_PRIORITY=1 and an email message has an RFC 822 `Priority:` header line, then the priority specified in that header line will instead be used to set the priority of the resulting SMS message. Specifically, if `USE_HEADER_PRIORITY=0`, then the SMS priority flag is always set in accord with the `DEFAULT_PRIORITY` option and the RFC 822 `Priority:` header line is always ignored. If `USE_HEADER_PRIORITY=1`, then the originating email message's RFC 822 `Priority:` header line is used to set the SMS message's priority flag. If that header line is not present, then the SMS priority flag is set using the `DEFAULT_PRIORITY` option.

The mapping used to translate RFC 822 `Priority:` header line values to SMS priority flags is shown in table that follows:

Table C-11 Mapping for Translating `Priority` Header to SMS Priority Flags

RFC 822	SMS priority flag		
Priority: value	GSM	TDMA	CDMA
Third	Non-priority (0)	Bulk (0)	Normal (0)
Second	Non-priority (0)	Bulk (0)	Normal (0)
Non-urgent	Non-priority (0)	Bulk (0)	Normal (0)
Normal	Non-priority (0)	Normal (1)	Normal (0)
Urgent	Priority (1)	Urgent (2)	Urgent (2)

DEFAULT_PRIVACY

(*integer, -1, 0 - 255*) Whether or not to set the privacy flag in an SMS message, and what value to use is controlled with the `DEFAULT_PRIVACY` and USE_HEADER_SENSITIVITY options. By default, a value of -1 is used for `DEFAULT_PRIVACY`. Table C-12 that follows shows the result of setting the `DEFAULT_PRIVACY` and USE_HEADER_SENSITIVITY options to various values.

Table C-12 Result of Values for `DEFAULT_PRIVACY` and `USE_HEADER_SENSITIVITY`

DEFAULT_PRIVACY	USE_HEADER_SENSITIVITY	Result
-1	0	The SMS privacy flag is never set in SMS messages.
n >= 0	0	The SMS privacy flag is always set to the value n. RFC 822 `Sensitivity:` header lines are always ignored.
-1 (default)	1 (default)	The SMS message's privacy flag is only set when the originating email message has an RFC 822 `Sensitivity:` header line. In that case, the SMS privacy flag is set to correspond to the `Sensitivity:` header line's value. This is the default.
n >= 0	1	The SMS message's privacy flag is set to correspond to the originating email message's RFC 822 `Sensitivity:` header line. If the email message does not have a `Sensitivity:` header line, then the value of the SMS privacy flag is set to n.

The SMS interpretation of privacy values is shown in Table C-13 that follows:

Table C-13 SMS Interpretation of Privacy Values

Value	Description
0	Unrestricted
1	Restricted
2	Confidential
3	Secret
>= 4	Undefined

The mapping used to translate RFC 822 `Sensitivity:` header line values to SMS privacy values is shown in Table C-14 that follows:

Table C-14 Mapping Translation of `Sensitivity` Headers to SMS Privacy Values

RFC 822 Sensitivity: value	SMS privacy value
Personal	1 (Restricted)
Private	2 (Confidential)
Company confidential	3 (Secret)

DEFAULT_SERVICE_TYPE

(*string, 0 - 5 bytes*) Service type to associate with SMS messages generated by the channel. By default, no service type is specified (that is, a zero length string). Some common service types are: `CMT` (cellular messaging), `CPT` (cellular paging), `VMN` (voice mail notification), `VMA` (voice mail alerting), `WAP` (wireless application protocol), and `USSD` (unstructured supplementary data services).

DEFAULT_SOURCE_ADDRESS

(*string, 0 - 20 bytes*) Source address to use for SMS messages generated from email messages. Note that the value specified with this option is overridden by the email message's originator address when `USE_HEADER_FROM=1`. By default, the value is disabled, that is, has a value of `0`.

DEFAULT_SOURCE_NPI

(*integer, 0 - 255*) By default, source addresses will be assigned an NPI value of zero. With this option, an alternate integer value in the range 0 to 255 may be assigned. See the description of the DEFAULT_DESTINATION_NPI option for a table of typical NPI values.

DEFAULT_SOURCE_TON

(*integer, 0 - 255*) By default, source addresses will be assigned a TON designator value of zero. With this option, an alternate integer value in the range 0 to 255 may be assigned. See the description of the DEFAULT_DESTINATION_TON option for a table of typical TON values.

DEFAULT_VALIDITY_PERIOD

(*string, 0 - 252 bytes*) By default, SMS messages are not given a relative validity period; instead, they use the SMSC's default value. Use this option to specify a different relative validity period. Values may be specified in units of seconds, minutes, hours, or days. Table C-15 that follows specifies the format and description of the various values for this option:

Table C-15 `DEFAULT_VALIDITY_PERIOD` Format and Values

Format	Description
nnn	Implicit units of seconds; for example, 604800
*nnn*s	Units of seconds; for example, 604800s
*nnn*m	Units of minutes; for example, 10080m
*nnn*h	Units of hours; for example, 168h
*nnn*d	Units of days; for example, 7d

A specification of 0, 0s, 0m, 0h, or 0d may be used to select the SMSC's default validity period. That is, when a specification of 0, 0s, 0m, 0h, or 0d is used, an empty string is specified for the validity period in generated SMS messages.

Note that this option does not accept values in UTC format.

DESTINATION_ADDRESS_NUMERIC

(0 or 1) Use this option to strip all non-numeric characters from the SMS destination address extracted from the email envelope `To:` address. For instance, if the envelope `To:` address is:

```
"(800) 555-1212"@sms.siroe.com
```

then it will be reduced to:

```
8005551212@sms.siroe.com
```

To enable this stripping, specify a value of 1 for this option. By default, this stripping is disabled which corresponds to an option value of 0. Note that when enabled, the stripping is done before any destination address prefix is added via the DESTINATION_ADDRESS_PREFIX option.

DESTINATION_ADDRESS_PREFIX

(*string*) In some instances, it may be necessary to ensure that all SMS destination addresses are prefixed with a fixed text string; for example, "+". This option may be used to specify just such a prefix. The prefix will then be added to any SMS destination address which lacks the specified prefix. To prevent being stripped by the DESTINATION_ADDRESS_NUMERIC option, this option is applied after the `DESTINATION_ADDRESS_NUMERIC` option.

PROFILE

(*string*) Specify the SMS profiling to be used with the SMSC. Possible values are `GSM`, `TDMA`, and `CDMA`. When not specified, GSM is assumed. This option is only used to select defaults for other channel options such as DEFAULT_PRIORITY and DEFAULT_PRIVACY.

USE_SAR

(0 or 1) Sufficiently large email messages may need to be broken into multiple SMS messages. When this occurs, the individual SMS messages can optionally have sequencing information added using the SMS sar_ fields. This produces a "segmented" SMS message which can be re-assembled into a single SMS message by the receiving terminal. Specify `USE_SAR=1` to indicate that this sequencing information is to be added when applicable. The default is to not add sequencing information and corresponds to `USE_SAR=0`.

When `USE_SAR=1` is specified, the REVERSE_ORDER option is ignored.

C.3.3.4 SMPP Options

The following options allow for specification of SMPP protocol parameters. The options with names beginning with the string "ESME_" serve to identify the MTA when it acts as an External Short Message Entity (ESME); that is, when the MTA binds to an SMPP server in order to submit SMS messages to the server's associated SMSC.

ESME_ADDRESS_NPI

(integer, 0 - 255) By default, bind operations will specify an ESME NPI value of zero indicating an unknown NPI. With this option, an alternate integer value in the range 0 to 255 may be assigned. See the description of the DEFAULT_DESTINATION_NPI option for a table of typical NPI values.

ESME_ADDRESS_TON

(integer, 0 - 255) By default, bind operations will specify an ESME TON value of 0. With this option, an alternate integer value in the range 0 to 255 may be assigned. See the description of the DEFAULT_DESTINATION_TON option for a table of typical TON values.

ESME_IP_ADDRESS

(*string, 0 - 15 bytes*) When binding to the SMPP server, the BIND PDU indicates that the client's (that is, ESME's) address range is an IP address. This is done by specifying a TON of 0x00 and an NPI of 0x0d. The value of the address range field is then set to be the IP address of the host running the SMS channel. Specify the IP address in dotted decimal format; for example, 127.0.0.1.

ESME_PASSWORD

(*string, 0 - 8 bytes*) When binding to the SMPP server, a password may be required. If so, then specify that password with this option. By default, a zero-length password string is presented.

ESME_SYSTEM_ID

(*string, 0 - 15 bytes*) When binding to the SMPP server, a system ID for the MTA may be supplied. By default, no system ID is specified (that is, a zero-length string is used). To specify a system ID, use this option.

ESME_SYSTEM_TYPE

(*string, 0 - 12 bytes*) When binding to the SMPP server, a system type for the MTA may be supplied. By default, no system type is specified (that is, a zero-length string is used).

MAX_PAGES_PER_BIND

(*integer, >= 0*) Some SMPP servers may limit the maximum number of SMS messages submitted during a single, bound session. In recognition of this, this option allows specification of the maximum number of SMS messages to submit during a single session. Once that limit is reached, the channel will unbind, close the TCP/IP connection, re-connect, and then rebind.

By default, a value of 1024 is used for MAX_PAGES_PER_BIND. Note that the channel will also detect ESME_RTHROTTLED errors and adjust MAX_PAGES_PER_BIND during a single run of the channel accordingly.

REVERSE_ORDER

(*0 or 1*) When an email message generates more than one SMS message, those SMS messages can be submitted to the SMSC in sequential order (REVERSE_ORDER=0), or reverse sequential order (REVERSE_ORDER=1). Reverse sequential order is useful for situations where the receiving terminal displays the last received message first. In such a case, the last received message will be the first part of the email message rather than the last. By default, REVERSE_ORDER=1 is used.

Note that this option is ignored when USE_SAR=1 is specified.

SMPP_MAX_CONNECTIONS

(*integer, 1 - 50*) This option controls the maximum number of simultaneous SMPP connections per process. As each connection has an associated thread, this option also places a limit on the maximum number of "worker" threads per process. By default, `SMPP_MAX_CONNECTIONS=20`.

SMPP_PORT

(*integer, 1 - 65535*) The TCP port which the SMPP server listens on may be specified with either this option or the `port` channel keyword. This port number must be specified through either of these two mechanisms. If it is specified with both mechanisms, then the setting made with the `SMPP_PORT` option takes precedence. Note that there is no default value for this option.

For two-way SMS, make sure its the same port as the `LISTEN_PORT` for the SMPP relay.

SMPP_SERVER

(*string, 1 - 252 bytes*) For one-way SMS, by default, the IP host name of the SMPP server to connect to is the official host name associated with the channel; that is, the host name shown on the second line of the channel's definition in MTA's configuration. This option may be used to specify a different host name or IP address which will override that specified in the channel definition. When specifying an IP address, use dotted decimal notation; for example, 127.0.0.1.

For two-way SMS, set to point to the host name or IP address of the SMS Gateway Server. If using the SMPP relay's `LISTEN_INTERFACE_ADDRESS` option, then be sure to use the host name or IP address associated with the specified network interface address.

TIMEOUT

(*integer, >= 2*) By default, a timeout of 30 seconds is used when waiting for data writes to the SMPP server to complete or for data to be received from the SMPP server. Use the `TIMEOUT` option to specify, in units of seconds, a different timeout value. The specified value should be at least 1second.

C.3.3.5 Localization Options

In constructing SMS messages, the SMS channel has a number of fixed text strings it puts into those messages. These strings, for example, introduce the email's `From:` address and `Subject:` header line. With the channel options described in this section, versions of these strings may be specified for different languages and a default language for the channel then specified. Example C-2 shows the language part of the option file:

Example C-2 Language Specification Part of Channel Option File

```
LANGUAGE=_default-language_

[language=i-default]
FROM_PREFIX=From:
SUBJECT_PREFIX=Subj:
CONTENT_PREFIX=Msg:
LINE_STOP= NO_MESSAGE=[no message]
REPLY_PREFIX=Re:

[language=en]
FROM_PREFIX=From:
SUBJECT_PREFIX=Subj:
CONTENT_PREFIX=Msg:
LINE_STOP=
NO_MESSAGE=[no message]
REPLY_PREFIX=Re:
  ...
```

Within each `[language=x]` block, the localization options relevant to that language may be specified. If a particular option is not specified within the block, then the global value for that option is used. A localization option specified outside of a{{ [language=x]}} block sets the global value for that option.

For the options listed below, the string values must be specified using either the US-ASCII or UTF-8 character sets. Note that the US-ASCII character set is a special case of the UTF-8 character set.

CONTENT_PREFIX

(*string, 0 - 252 bytes*) Text string to place in the SMS message before the content of the email message itself. Default global value is the US-ASCII string "`Msg:`".

DSN_DELAYED_FORMAT

(string, 0-256 characters) Formatting string for delivery delay notifications. By default, an empty string is used for this option, thereby inhibiting the conversion to SMS of delay notifications. Note that GATEWAY_NOTIFICATIONS must be set to `1` for this option to be in effect. This option is ignored when `GATEWAY_NOTIFICATIONS=0`.

DSN_FAILED_FORMAT

(string, 0-256 characters) Formatting string for permanent delivery failure notifications. The default value of this option is the string:

```
Unable to deliver your message to $a; no further delivery attempts will be
made.
```

To inhibit conversion of failure notifications, specify an empty string for this option. Note that GATEWAY_NOTIFICATIONS must be set to `1` for this option to be in effect. This option is ignored when `GATEWAY_NOTIFICATIONS=0`.

DSN_RELAYED_FORMAT

(string, 0-256 characters) Formatting string for relay notifications. The default value is the string:

```
Your message to $a has been relayed to a messaging system which may not
provide a final delivery confirmation
```

To inhibit conversion of relay notifications, specify an empty string for this option. Note that GATEWAY_NOTIFICATIONS must be set to `1` for this option to be in effect. This option is ignored when `GATEWAY_NOTIFICATIONS=0`.

DSN_SUCCESS_FORMAT

(string, 0-256 characters) Formatting string for successful delivery notifications. The default value is the string:

```
Your message to $a has been delivered
```

To inhibit conversion of successful delivery notifications, specify an empty string for this option. Note that GATEWAY_NOTIFICATIONS must be set to `1` for this option to be in effect. This option is ignored when `GATEWAY_NOTIFICATIONS=0`.

FROM_FORMAT

(*string, 0 - 252 bytes*) Formatting template to format the originator information to insert into the SMS message. The default global value is the US-ASCII string "$a" which substitutes in the originator's email address. See C.3.3.6 Formatting Templates

FROM_NONE

(*string, 0 - 252 bytes*) Text string to place in the SMS message when there is no originator address to display. The default global value is an empty string.

Note that normally, this option will never be used as sites will typically reject email messages which lack any originator address.

LANGUAGE

(*string, 0 - 40 bytes*) The default language group to select text strings from. If not specified, then the language will be derived from the host's default locale specification. If the host's locale specification is not available or corresponds to "C", then i-default will be used. (i-default corresponds to "English text intended for an international audience.")

LINE_STOP

(*string, 0 - 252 bytes*) Text string to place in the SMS message between lines extracted from the email message. The default global value is the US-ASCII space character, " ".

NO_MESSAGE

(*string, 0 - 252 bytes*) Text string to place in the SMS message to indicate that the email message had no content. The default global value is the US-ASCII string `"[no message]"`.

SUBJECT_FORMAT

(*string, 0 - 252 bytes*) Formatting template to format the content of the `Subject:` header line for display in the SMS message. The global default value for this option is the US-ASCII string `"($s)"`. See C.3.3.6 Formatting Templates for further details.

See the SUBJECT_NONE option for a description of the handling when there is no Subject: header line or the content of that header line is an empty string.

SUBJECT_NONE

(*string, 0 - 252 bytes*) Text string to display when the originating email message either has no Subject: header line, or the Subject: header line's value is an empty string. The default global value for this option is the empty string.

DEBUG

(integer, bitmask) Enable debug output. The default value is 6 which selects warning and error messages. Any non-zero value enables debug output for the channel itself, the same as specifying master_debug on the channel definition.Table C-16 defines the bit values of the DEBUG bitmask.

Table C-16 DEBUG Bitmask

Bit	Value	Description
0-31	-1	Extremely verbose output
0	1	Informational messages
1	2	Warning messages
3	4	Error messages
3	8	Subroutine call tracing
4	16	Hash table diagnostics
5	32	I/O diagnostics, receive
6	64	I/O diagnostics, transmit
7	128	SMS to email conversion diagnostics (mobile originate and SMS notification)
8	256	PDU diagnostics, header data
9	512	PDU diagnostics, body data
10	1024	PDU diagnostics, type-length-value data
11	2048	Option processing; sends all option settings to the log file.

C.3.3.6 Formatting Templates

The formatting templates specified with the FROM_FORMAT, SUBJECT_FORMAT, and all the DSN_* channel options are UTF-8 strings which may contain a combination of literal text and substitution sequences. Assuming the sample email address of

```
Jane Doe <user@siroe>
```

The recognized substitution sequences are shown in Table C-17 that follows:

Table C-17 Substitution Sequences

Sequence	Description
$a	Replace with the local and domain part of the originator's email address (for example, "user@siroe")
$d	Replace with the domain part of the originator's email address (for example, "domain")
$p	Replace with the phrase part, if any, of the originator's email address (for example, "Jane Doe")
$s	Replace with the content of the Subject: header line
$u	Replace with the local part of the originator's email address (for example, "user")
\x	Replace with the literal character "x"

For example, the formatting template

```
From: $a
```

produces the text string

```
From: user@siroe
```

The construct,

```
${xy:alternate text}
```

may be used to substitute in the text associated with the sequence x. If that text is the empty string, the text associated with the sequence y is instead used. And, if that text is the empty string, to then substitute in the alternate text. For example, consider the formatting template

```
From: ${pa:unknown sender}
```

For the originator email address

```
John Doe <jdoe@siroe.com>
```

which has a phrase part, the template produces:

```
From: John Doe
```

However, for the address

```
jdoe@siroe.com
```

which lacks a phrase, it produces

```
From: jdoe@siroe.com
```

And for an empty originator address, it produces

```
From: unknown sender
```

C.3.4 Adding Additional SMS Channels

You may configure the MTA to have more than one SMS channel. There are two typical reasons to do this:

1. To communicate with different SMPP servers.

This is quite straightforward: just add an additional SMS channel to the configuration, being sure to (a) give it a different channel name, and (b) associate different host names with it. For example,

```
sms_mway port 55555 threaddepth 20
smpp.siroe.com

sms_ace port 777 threaddepth 20
sms.ace.net
```

Note that no new rewrite rule is needed. If there is no directly matching rewrite rule, Messaging Sever looks for a channel with the associated host name. For example, if the server is presented with `user@host.domain`, it would look for a channel of the name "host.domain". If it finds such a channel, it routes the message there. Otherwise, it starts looking for a rewrite rule for the ".domain" and if none is there, then for the dot (".") rule. For more information on rewrite rules, see Configuring Rewrite Rules.

2. To communicate with the same SMPP server but using different channel options.
   To communicate with the same SMPP server, using different channel options, specify the same SMPP server in the SMPP_SERVER channel option for each channel definition.
   Using this mechanism is necessary since two different channels cannot have the same official host name (that is, the host name listed in the second line of the channel definition). To allow them to communicate with the same SMPP server, define two separate channels, with each specifying the same SMPP server in their SMPP_SERVER in their channel option files.
   For example, you could have the following channel definitions,

```
sms_mway_1 port 55555 threaddepth 20
SMS-DAEMON-1

sms_mway_2 port 55555 threaddepth 20
SMS-DAEMON-2
```

   and rewrite rules,

```
sms-1.siroe.com $u%sms-1.siroe.com@SMS-DAEMON-1
sms-2.siroe.com $U%sms-2.siroe.com@SMS-DAEMON-2
```

   Then, to have them both use the same SMPP server, each of these two channels would specify SMPP_SERVER=smpp.siroe.com in their channel option file.

C.3.5 Adjusting the Frequency of Delivery Retries

When an SMS message cannot be delivered owing to temporary errors (for example, the SMPP server is not reachable), the email message is left in the delivery queue and retried again later. Unless configured otherwise, the Job Controller will not re-attempt delivery for an hour. For SMS messaging, that is likely too long to wait. As such, it is recommended that the backoff channel keyword be used with the SMS channel to specify a more aggressive schedule for delivery attempts. For example,

```
sms_mway port 55555 threaddepth 20 \
  backoff "pt2m" "pt5m" "pt10m" "pt30m" notices 1
smpp.siroe.com
```

With the above settings, a redelivery attempt will be made at two minutes after the first attempt. If that then fails, then five minutes after the second attempt. Then ten minutes later and finally every thirty minutes. The `notices 1` channel keyword causes the message to be returned as undeliverable if it cannot be delivered after a day.

C.3.6 Sample One-Way Configuration (MobileWay)

The MTA SMS channel may be used with any SMPP V3.4 compatible SMPP server. For purposes of illustrating an example configuration, this section explains how to configure the SMS channel for use with a MobileWay SMPP server. MobileWay (http://www.mobilway.com) is a leading provider of global data and SMS connectivity. By routing your SMS traffic through MobileWay, you can reach SMS subscribers on most of the major SMS networks throughout the world.

When requesting an SMPP account with MobileWay, you may be asked to answer the following questions:

- IP address of your SMPP client: Supply the IP address of your Messaging Server system as seen by other domains on the Internet.
- Default validity period: This is the SMS validity period which MobileWay will use should a validity period not be specified in the SMS messages you submit. SMS messages which cannot be delivered before this validity period expires will be discarded. Supply a reasonable value (for example, 2 days, 7 days, etc.).
- Window size: This is the maximum number of SMS messages your SMPP client will submit before it will stop and wait for responses from the SMPP server before submitting any further SMS messages. You must supply a value of 1 message.
- Timezone: Specify the timezone in which your Messaging Server system operates. The timezone should be specified as an offset from GMT.
- Timeout: Not relevant to one-way SMS messaging.
- IP address and TCP port for outbind requests: Not relevant for one-way SMS messaging.

After supplying MobileWay with the answers to the above questions, they will provide you with an SMPP account and information necessary to communicate with their SMPP servers. This information includes

```
Account Address: a.b.c.d:p
Account Login: system-id
Account Passwd: secret
```

The Account Address field is the IP address, `a.b.c.d`, and TCP port number, P., of the MobileWay SMPP server you will be connecting to. Use these values for the SMPP_SERVER and SMPP_PORT channel options. The Account Login and Passwd are, respectively, the values to use for the ESME_SYSTEM_ID and ESME_PASSWORD channel options. Using this information, your channel's option file should include

```
SMPP_SERVER=a.b.c.d
SMPP_PORT=p
ESME_SYSTEM_ID=system-id
ESME_PASSWORD=secret
```

Now, to interoperate with MobileWay you need to make two additional option settings

```
ESME_ADDRESS_TON=0x01
DEFAULT_DESTINATION_TON=0x01
```

The rewrite rule in the `imta.cnf` file can appear as

```
sms.your-domain $u@sms.your-domain
```

And, the channel definition in the `imta.cnf` file can appear as

```
sms_mobileway
sms.your-domain
```

Once the channel option file, rewrite rule, and channel definition are in place, a test message may be sent. MobileWay requires International addressing of the form

`+<country-code><subscriber-number>`

For instance, to send a test message to the North American subscriber with the subscriber number (800) 555-1212, you would address your email message to

`+18005551212@sms.your-domain`

C.3.6.1 Debugging

To debug the channel, specify the `master_debug` channel keyword in the channel's definition. For example,

```
sms_mway port 55555 threaddepth 20 \
backoff "pt2m" "pt5m" "pt10m" "pt30m" notices 1 master_debug
```

With the `master_debug` channel keyword, basic diagnostic information about the channel's operation will be output to the channel's log file. For verbose diagnostic information about the SMPP transactions undertaken by the channel, also specify

`DEBUG=-1`

in the channel's option file.

C.3.7 Configuring the SMS Channel for Two-Way SMS

For general directions on configuring the SMS channel, refer to earlier topics starting with C.3 SMS Channel Configuration. Configure the SMS channel as though it will be talking directly to the remote SMSC, with the exceptions listed in Table C-18 that follows:

Table C-18 Two-Way Configuration Exceptions

Exception	Explanation
`master` channel keyword	Remove the `master` channel keyword, if present.It is no longer needed for SMS channel configuration.
`SMPP_SERVER`	Set to point to the host name of IP address of the SMS Gateway Server. If using the SMPP relay's `LISTEN_INTERFACE_ADDRESS` option (see C.5.7 Configuration Options), then be sure to use the host name or IP address associated with the specified network interface address.
`SMPP_PORT`	Same TCP port as used for the `LISTEN_PORT` setting used to instantiate the SMPP relay (see C.5.5.2 An SMPP Relay
`DEFAULT_SOURCE_ADDRESS`	Pick a value and then configure the remote SMSC to route this address back to the Gateway SMPP server. In the SMS channel's option file, specify the chosen value with this option.
`GATEWAY_PROFILE`	Set to match the gateway profile name. See C.5.5.1 A Gateway Profile
`USE_HEADER_FROM`	Set to `0`.

All other channel configurations should be done as described in the SMS Channel documentation.

As mentioned in C.5.1 Setting Up Bidirectional SMS Routing, the remote SMSC needs to be configured to route the SMS address, defined in the `DEFAULT_SOURCE_ADDRESS` channel option, to the Gateway's SMPP server using the TCP port number specified with the `LISTEN_PORT` option. (For how to specify the `LISTEN_PORT`, see C.5.5.3 An SMPP Server.)

Note that multiple SMS channels may use the same SMPP relay. Similarly, there need be only one SMPP server or gateway profile to handle SMS replies and notifications for multiple SMS channels. The ability to configure multiple relays, servers, and gateway profiles exists to effect different usage characteristics through configuration options.

C.4 SMS Gateway Server Theory of Operation

The SMS Gateway Server facilitates two-way SMS through mechanisms that allow mobile originated SMS messages to be matched to the correct email address. The following SMS Gateway Server topics are covered in this section:

- C.1 Introduction
- C.2 SMS Channel Theory of Operation
- C.3 SMS Channel Configuration
- C.4 SMS Gateway Server Theory of Operation
- C.5 SMS Gateway Server Configuration
- C.6 SMS Gateway Server Storage Requirements
- C.7 SMS Configuration Examples

C.4.1 Function of the SMS Gateway Server

The SMS Gateway Server simultaneously functions as both an SMPP relay and server. It may be configured to have multiple "instantiations" of each function. For instance, it may be configured to have three different SMPP relays, each listening on different TCP ports or network interfaces and relaying to different remote SMPP servers. Similarly, it may be configured to have four different SMPP servers, each listening on different combinations of TCP ports and network interfaces.

The SMS Gateway Server may be configured with zero or more gateway profiles for sending SMS messages to email. Each gateway profile describes which destination SMS addresses match the profile,

how to extract the destination email addresses from SMS messages, and various characteristics of the SMS to email conversion process. Each SMS message presented to the SMS Gateway Server through either its SMPP relay or server are compared to each profile. If a match is found, then the message is routed to email.

Finally, the gateway profiles also describe how to handle notification messages returned by remote SMSCs in response to previous email-to-mobile messages.

C.4.2 Behavior of the SMPP Relay and Server

When acting as an SMPP relay, the SMS Gateway Server attempts to be as transparent as possible, relaying all requests from local SMPP clients on to a remote SMPP server and then relaying back the remote server's responses. There are two exceptions:

- When a local SMPP client submits a message whose SMS destination address matches one of the configured gateway profiles, the submitted SMS message is sent directly back to email; the SMS message is not relayed to a remote SMPP server.
- When a local or remote SMPP client submits a message whose SMS destination address matches a unique SMS source address previously generated by the SMPP relay, the SMS message is a reply to a previously relayed message. This reply is directed back to the originator of the original message.

Note that typically the SMS Gateway Server will be configured such that the unique SMS source addresses which it generates match one of the gateway profiles.

> **Note -**
> The SMS Gateway Server's SMPP relay is only intended for use with qualified, SMPP clients, that is, the Messaging Server's SMS channel. It is not intended for use with arbitrary SMPP clients.

When acting as an SMPP server, the SMS Gateway Server directs SMS messages to email for three circumstances:

- The SMS messages are mobile originated and match a gateway profile.
- The SMS messages are mobile originated and the SMS destination address matches a previously generated unique SMS source address.
- The SMS messages are SMS notifications which correspond to email-to-mobile messages previously relayed by the SMS Gateway Server's SMPP relay.

All other SMS messages are rejected by the SMPP server.

C.4.3 Remote SMPP to Gateway SMPP Communication

Remote SMPP clients communicate to the Gateway SMPP server with Protocol Data Units (PDUs). Remote SMPP clients emit request PDUs to which the Gateway SMPP server responds. The Gateway SMPP server operates synchronously. It completes the response to a request PDU before it processes the next request PDU from the connected remote SMPP client.

Table C-19 that follows lists the request PDUs the Gateway SMPP server handles, and specifies the Gateway SMPP server's response.

Table C-19 SMPP Server Protocol Data Units

Request PDU	SMPP Server Response
BIND_TRANSMITTERBIND_TRANSCEIVERUNBIND	Responded to with the appropriate response PDU. Authentication credentials are ignored.
OUTBIND	Gateway SMPP server sends back a BIND_RECEIVER PDU. Authentication credentials presented are ignored.
SUBMIT_SMDATA_SM	Attempts to match the destination SMS address with either a unique SMS source address or the SELECT_RE setting of a Gateway profile. If neither is matched, the PDU is rejected with an ESME_RINVDSTADR error.
DELIVER_SM	Attempts to find either the destination SMS address or the receipted message ID in the historical record. If neither is matched, returns the error ESME_RINVMSGID.
BIND_RECEIVER	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
SUBMIT_MULTI	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
REPLACE_SM	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
CANCEL_SM	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
QUERY_SM	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
QUERY_LAST_MSGS	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
QUERY_MSG_DETAILS	Not supported. Returns a GENERIC_NAK PDU with an ESME_RINVCMDID error.
ENQUIRE_LINK	Returns ENQUIRE_LINK_RESP PDU.
ALERT_NOTIFICATION	Accepted but ignored.

C.4.4 SMS Reply and Notification Handling

The SMS Gateway Server maintains a historical record of each SMS message relayed through its SMPP relays. The need to use historical data arises from the fact that when submitting an email message to SMS it is generally not possible to convert the email address of the message's originator to an SMS source address. Since any SMS replies and notifications will be directed to this SMS source address, a problem arises. This is resolved by using automatically generated, unique SMS source addresses in relayed messages. The remote SMSCs are then configured to route these SMS source addresses back to the Gateway SMPP server.

The historical data is represented as an in-memory hash table of message IDs and generated, unique SMS source addresses. This data along with the associated email origination data are also stored on disk. The disk based storage is a series of files, each file representing HASH_FILE_ROLLOVER_PERIOD

seconds of transactions (the default is 30 minutes). Each file is retained for RECORD_LIFETIME seconds (the default is 3 days). See http://docs.sun.com/doc/819-4439 for a discussion of the in-memory and on-disk resource requirements of the historical data.

Each record has three components:

- Email origination data (such as, envelope From: and To: addresses). This data is supplied by the MTA SMS channel when it submits a message.
- The unique SMS source address generated by the SMPP relay and inserted into the relayed SMS message.
- The resulting receipted message ID returned by the remote SMSC's SMPP server when it accepts a submission.

C.4.4.1 Routing Process for SMS Replies

The Gateway SMPP relays and servers use historical records to handle SMS replies, notifications and mobile originated messages. When an SMS message is presented to the SMPP relay or server, the following routing process is followed:

1. The SMS destination address is compared against the historical record to see if there is a matching, unique SMS source address that the SMPP relay previously generated. If a match is found, see Step 6. .
2. If there is no match, but the message is an SMS notification (SMPP DELIVER_SM PDU), then the receipted message ID, if present, is compared against the historical record. If a match is found, go to Step 8. The SMS Gateway Server actually allows these to be presented to either the SMPP relay or SMPP server.
3. If there is no match, then the destination SMS address is compared against the SELECT_RE option expressions for each configured gateway profile. If a match is found, then go to Step 9.
4. If there is no match and the SMS message was presented to the Gateway SMPP relay, then the message is relayed to the remote SMPP server.
5. If there is no match and the SMS message was presented to the Gateway SMPP server, then the message is determined to be an invalid message and an error response is returned in the SMPP response PDU. For email to SMS, a Non Delivery Notification (NDN) is eventually generated.
6. If a matching unique SMS source address was found, then the SMS message is further inspected to see if it is a reply or a notification message. To be a notification message it must be a SUBMIT_SM PDU with a receipted message ID. Otherwise, it is considered to be a reply.
7. If it is a reply then the SMS message is converted to an email message using the origination email information from the historical record.
8. If it is a notification, then the SMS message is converted to an email Delivery Status Notification (DSN) as per RFC 1892-1894. Note that the ESMTP NOTIFY flags (RFC 1891) of the original email message will be honored (For example, if the SMS message is a "success" DSN but the original email message requested only "failure" notifications, then the SMS notification will be discarded).
9. If the destination SMS address matches a SELECT_RE option in a configured gateway profile, then the SMS message is treated as a mobile originated message and converted back to email message as per the PARSE_RE_n rules for that gateway profile. If the conversion fails, then the SMS message is invalid and an error response is returned.

C.5 SMS Gateway Server Configuration

This section gives directions on how to set up the SMS Gateway Server for both email-to-mobile and mobile-to-email functionality. This section includes the following topics:

- C.6 SMS Gateway Server Storage Requirements
- C.7 SMS Configuration Examples

C.5.1 Setting Up Bidirectional SMS Routing

The recommended way to set up bidirectional email and SMS routing between the MTA and SMSC is a three step process:

- C.5.1.1 Set the SMS Address Prefix-- Choose an SMS address prefix. Any prefix may be used, so long as it is ten characters or less.
- C.5.1.2 Set the Gateway Profile-- Reserve the prefix for use with the SMS Gateway Server (by setting the gateway profile).
- C.5.1.3 Configure the SMSC-- Configure the SMSC to route SMS destination addresses to the SMS Gateway SMPP server that start with the prefix. Mobile originated email will have only the prefix. Replies and notifications will have the prefix followed by exactly ten decimal digits.

C.5.1.1 Set the SMS Address Prefix

The source SMS addresses generated by the MTA SMS channel should be set to match the selected SMS address prefix. Do this by setting the following:

- MTA SMS channel options:
  `USE_HEADER_FROM=0`
  `DEFAULT_SOURCE_ADDRESS=`*prefix*
  The first setting causes the channel to not attempt to set the SMS source address from information contained in the email message. The second setting causes the SMS source address to be set (to the selected prefix) when it is not set from any other source.
- Recognize the prefix as an SMS destination address to accept and route to email. Do this by specifying the `SELECT_RE` gateway profile option as follows:
  `SELECT_RE=`*prefix*

C.5.1.2 Set the Gateway Profile

The SMS Gateway Server's gateway profile should then be set to make all relayed SMS source addresses unique. This is the default setting but may be explicitly set by specifying the gateway profile option `MAKE_SOURCE_ADDRESSES_UNIQUE=1`. This will result in relayed SMS source addresses of the form:

```
prefixnnnnnnnnnn
```

where nnnnnnnnnn will be a unique, ten digit decimal number.

C.5.1.3 Configure the SMSC

Finally, the SMSC should be configured to route all SMS destination addresses matching the prefix (either just the prefix, or the prefix plus a ten digit number) to the SMS Gateway Server's SMPP server. The regular expression for such a routing will be similar to:

```
prefix([0-9]{10,10}){0,1}
```

where `prefix` is the value of `DEFAULT_SOURCE_ADDRESS`, [0-9] specifies the allowed values for the ten digit number, {10, 10} specifies that there will be a minimum of ten digits and a maximum of ten digits, and {0, 1} specifies that there can be zero or one of the 10-digit numbers.

C.5.2 Enabling and Disabling the SMS Gateway Server

- To enable the SMS Gateway Server, the configutil parameter `local.smsgateway.enable` must be set to the value `1`. Use the following configuration utility command to set it:
  `# configutil -o local.smsgateway.enable -v 1`
- To disable the gateway server, set `local.smsgateway.enable` to the value `0`, using the following command:
  `# configutil -o local.smsgateway.enable -v 0`

C.5.3 Starting and Stopping the SMS Gateway Server

After the SMS Gateway Server is enabled, it may be started and stopped with the following commands:

```
# start-msg sms
```

and

```
# stop-msg sms
```

C.5.4 SMS Gateway Server Configuration File

In order to operate, the SMS Gateway Server requires a configuration file. The configuration file is a Unicode text file encoded using UTF-8. The file can be an ASCII text file. The name of the file must be:

```
installation-directory/config/sms_gateway.cnf
```

Each option setting in the file has the following format:

```
option-name=option-value
```

Options that are part of an option group appear in the following format:

```
[group-type=group-name]
option-name-1=option-value-1
option-name-2=option-value-2
...
option-name-n=option-value-n
```

C.5.5 Configuring Email-To-Mobile on the Gateway Server

To implement the email-to-mobile part of two-way SMS, you must configure the following:

- C.5.5.1 A Gateway Profile
- C.5.5.2 An SMPP Relay
- C.5.5.3 An SMPP Server

C.5.5.1 A Gateway Profile

To configure an email-to-mobile gateway profile, follow these steps:

To Configure an Email-to-mobile Gateway Profile

1. Add a gateway profile to the SMS Gateway Server configuration file.

To add an option group, use the following format:

```
[GATEWAY_PROFILE=profile_name]
option-name-1=option-value-1
option-name-2=option-value-2a
...
option-name-n=option-value-n
```

The length of the gateway profile name, `profile_name` in the preceding format, must not exceed 11 bytes. The name must be the same as the name for the `GATEWAY_PROFILE` channel option in the SMS channel option file. The name is case insensitive. For a list of the valid channel options, see C.3.3 Available Options

2. Set the gateway profile options (for example, `SMSC_DEFAULT_CHARSET`) to match characteristics of the remote SMSC.
3. Set the other gateway profile options to match the SMS channel's email characteristics.
   A complete description of gateway profile options, see C.5.11 Gateway Profile Options
4. Set the `CHANNEL` option.
   Set its value to the name of the MTA SMS channel.
   When a notification is sent to email through the gateway, the resulting email message will be enqueued to the MTA using this channel name.

C.5.5.2 An SMPP Relay

To configure an SMPP Relay, complete the following steps:

To Configure an SMPP Relay

1. Add an SMPP relay instantiation (option group) to the SMS Gateway Server's configuration file.
   To add an option group, use the following format:

```
[SMPP_RELAY=relay_name]
option-name-1=option-value-1
option-name-2=option-value-2
...
option-name-n=option-value-n
```

   Any name may be used for the relay's name. All that matters is that the name not be used for any other SMPP relay instantiation within the same configuration file.
2. Set the `LISTEN_PORT` option.
   The value used for the SMS channel's `SMPP_PORT` option must match that used for the relay's `LISTEN_PORT` option. For the `LISTEN_PORT`, select a TCP port number which is not used by any other SMPP relay or server instantiation nor by any other server running on the same computer.
3. Set the `SERVER_HOST` option.
   The relay's `SERVER_HOST` option should give the host name for the remote SMSC's SMPP server. An IP address may be used in place of a host name.
4. Set the `SERVER_PORT` option.
   The relay's `SERVER_PORT` option should give the TCP port for the remote SMSC's SMPP server.
   For a complete description of all SMPP relay options, see C.5.9 SMPP Relay Options

C.5.5.3 An SMPP Server

To configure an SMPP server, complete the following steps:

To Configure an SMPP Server

1. Add an SMPP server instantiation (option group) to the SMS Gateway Server's configuration file.

To add an option group, use the following format:

```
[SMPP_SERVER=server_name]
option-name-1=option-value-1
option-name-2=option-value-2...
option-name-n=option-value-n
```

Any name may be used for the server's name. All that matters is that the name not be used for any other SMPP server instantiation within the same configuration file.

2. Set `LISTEN_PORT` option.
   Select a TCP port number which is not being used by any other server or relay instantiation. Additionally, the port number must not be being used by any other server on the same computer. The remote SMSC needs to be configured to route notifications via SMPP to the SMS Gateway Server system using this TCP port.
   For a complete description of all SMPP server options, see C.5.10 SMPP Server Options

C.5.6 Configuring Mobile-to-Email Operation

To configure mobile-to-email functionality, two configuration steps must be performed:

Note that multiple gateway profiles may use the same SMPP server instantiation. Indeed, the same SMPP server instantiation may be used for both email-to-mobile and mobile-to-email applications.

C.5.6.1 Configure a Mobile-to-Email Gateway Profile

For mobile origination, a gateway profile provides two key pieces of information: how to identify SMS messages intended for that profile and how to convert those messages to email messages. Note that this profile can be the same one used for email-to-mobile with the addition of the `SELECT_RE` option.

To configure the gateway profile, follow these steps:

To Configure the Gateway Profile

1. Add a gateway profile (option group) to the SMS Gateway Server's configuration file.
   To add an option group, use the following format:

```
[GATEWAY_PROFILE=profile_name]
option-name-1=option-value-1
option-name-2=option-value-2
...
option-name-n=option-value-n
```

   Any name of 11 characters or less may be used for the profile's name. All that matters is that it is not already used for another gateway profile within the same configuration file.
2. Set the `SELECT_RE` option must be specified for each gateway profile.
   The value of this option is an ASCII regular expression with which to compare SMS destination addresses. If an SMS destination address matches the regular expression, then the SMS message is sent through the gateway to email using the characteristics described by the matching profile.
   It is important to note that it is possible to configure multiple gateway profiles which have overlapping sets of SMS addresses (for example, a profile which matches the address 000 and

another which matches any other three-digit address). However, so doing should be avoided as an SMS message will be passed off to only one gateway profile: the first one which matches. And, moreover, the order in which they are compared is undefined.

3. Set the `CHANNEL` option.
   Its value should be the name of the MTA's SMS channel.
   For a complete description of all mobile origination options, see C.5.11 Gateway Profile Options

C.5.6.2 Configure a Mobile-to-Email SMPP Server

Adding an SMPP server is the same as for the email-to-mobile SMPP server (see C.5.5.3 An SMPP Server).

The remote SMSC needs to be configured to route SMS traffic to the gateway SMPP server. To do this, the SMS destination address used by the SMSC to route mobile-to-email traffic should be the value set for the gateway profile option `SELECT_RE`.

For example, if the SMS address 000 is to be used for mobile-to-email traffic, then the SMSC needs to be configured to route traffic for the SMS destination address 000 to the gateway SMPP server. The gateway profile should use the option setting `SELECT_RE=000`.

C.5.7 Configuration Options

The SMS Gateway Server configuration file options are detailed in this section. The tables that follow list all the available configuration options with a brief description of each. There is a table each for global options, SMPP relay options, SMPP server options, and SMS Gateway Server profile options.

In the subsections that follow, complete descriptions are given for all the available configuration options. The subsections are:

- C.5.8 Global Options
  Global options must be placed at the top of the configuration file, before any option groups. The remaining options must appear within option groups.
- C.5.9 SMPP Relay Options
- C.5.10 SMPP Server Options
- C.5.11 Gateway Profile Options

C.5.8 Global Options

The SMS Gateway Server presently has three categories of global options:

- C.5.8.1 Thread Tuning Options
- C.5.8.2 Historical Data Tuning
- C.5.8.3 Miscellaneous

All global options must be specified at the top of the configuration file, before any option groups are specified. Table C-20 lists all global configuration options.

Table C-20 Global Options

Options	Default	Description
DEBUG	6	Selects the types of diagnostic output generated
HISTORY_FILE_DIRECTORY	no default value	Absolute directory path for files of historical data
HISTORY_FILE_MODE	0770	Permissions for files of historical data
HISTORY_FILE_ROLLOVER_PERIOD	30 mins	Maximum length of time to write to the same file of historical data
LISTEN_CONNECTION_MAX	10,000	Maximum number of concurrent inbound connections across all SMPP relay and server instantiations
RECORD_LIFETIME	3 days	Lifetime of a record in the historical data archive
THREAD_COUNT_INITIAL	10 threads	Initial number of worker threads
THREAD_COUNT_MAXIMUM	50 threads	Maximum number of worker threads
THREAD_STACK_SIZE	64 Kb	Stack size for each worker thread

C.5.8.1 Thread Tuning Options

Each inbound TCP connection represents an SMPP session. The processing for a session is handled by a worker thread from a pool of threads. When the session processing needs to wait for completion of an I/O request, the worker thread parks the session and is given other work to perform. When the I/O request completes, the session is resumed by an available worker thread from the pool.

The following options allow for tuning of this pool of worker thread processes: THREAD_COUNT_INITIAL, THREAD_COUNT_MAXIMUM, THREAD_STACK_SIZE.

THREAD_COUNT_INITIAL

(integer, > 0_)_ Number of threads to initially create for the pool of worker threads. This count does not include the dedicated threads used to manage the in-memory historical data (2 threads) nor the dedicated threads used to listen for incoming TCP connections (one thread per TCP port/interface address pair the SMS Gateway Server listens on). The default value is for `THREAD_COUNT_INITIAL` is 10 threads.

THREAD_COUNT_MAXIMUM

(integer, >= `THREAD_COUNT_INITIAL`*)* Maximum number of threads to allow for the pool of worker threads. The default value is 50 threads.

THREAD_STACK_SIZE

(integer, > 0_)_ Stack size in bytes for each worker thread in the pool of worker threads. The default value is 65,536 bytes (64 Kb).

C.5.8.2 Historical Data Tuning

When an SMS message is relayed, the message ID generated by the receiving, remote SMPP server is saved in an in-memory hash table. Along with this message ID, information about the original email

message is also saved. Should that message ID subsequently be referenced by an SMS notification, this information may be retrieved. The retrieved information can then be used to send the SMS notification to the appropriate email recipient.

The in-memory hash table is backed to disk by a dedicated thread. The resulting disk files are referred to as "history files". These history files serve two purposes: to save, in nonvolatile form, the data necessary to restore the in-memory hash table after a restart of the SMS Gateway Server, and to conserve virtual memory by storing potentially lengthy data on disk. Each history file is only written to for `HASH_FILE_ROLLOVER_PERIOD` seconds after which time it is closed and a new history file created. When a history file exceeds an age of `RECORD_LIFETIME` seconds, it is deleted from disk.

The following options allow for tuning historical files: HISTORY_FILE_DIRECTORY, HISTORY_FILE_MODE, HISTORY_FILE_ROLLOVER_PERIOD, RECORD_LIFETIME.

HISTORY_FILE_DIRECTORY

(string, absolute directory path) Absolute path to the directory to which to write the history files. The directory path will be created if it does not exist. The default value for this option is:

msg-svr-base`/data/sms_gateway_cache/`

The directory used should be on a reasonably fast disk system and have more than sufficient free space for the anticipated storage; see C.6 SMS Gateway Server Storage Requirements to change this option to a more appropriate value.

HISTORY_FILE_MODE

(integer, octal value) File permissions to associated with the history files. By default, a value of 0770 (octal) is used.

HISTORY_FILE_ROLLOVER_PERIOD

(integer, seconds) The current history file is closed and a new one created every `HASH_FILE_ROLLOVER_PERIOD` seconds. By default, a value of 1800 seconds (30 minutes) is used.

RECORD_LIFETIME

(integer, seconds > 0_)_ Lifetime in seconds of a historical record. Records older than this lifetime will be purged from memory; history files older than this lifetime will be deleted from disk. By default, a value of 259,200 seconds (3 days) is used. Records stored in memory are purged in sweeps by a thread dedicated to managing the in-memory data. These sweeps occur every `HASH_FILE_ROLLOVER_PERIOD` seconds. Files on disk are purged when it becomes necessary to open a new history file.

C.5.8.3 Miscellaneous

These are miscellaneous options:

- C.1 Introduction
- C.2 SMS Channel Theory of Operation
- C.3 SMS Channel Configuration
- C.4 SMS Gateway Server Theory of Operation
- C.5 SMS Gateway Server Configuration
- C.6 SMS Gateway Server Storage Requirements
- C.7 SMS Configuration Examples

DEBUG

(integer, bitmask) Enable debug output. The default value is 6 which selects warning and error messages.

Table C-21 defines the bit values of the `DEBUG` bitmask.

Table C-21 DEBUG Bitmask

Bit	Value	Description
0-31	`-1`	Extremely verbose output
0	`1`	Informational messages
1	`2`	Warning messages
3	`4`	Error messages
3	`8`	Subroutine call tracing
4	`16`	Hash table diagnostics
5	`32`	I/O diagnostics, receive
6	`64`	I/O diagnostics, transmit
7	`128`	SMS to email conversion diagnostics (mobile originate and SMS notification)
8	`256`	PDU diagnostics, header data
9	`512`	PDU diagnostics, body data
10	`1024`	PDU diagnostics, type-length-value data
11	`2048`	Option processing; sends all option settings to the log file.

LISTEN_CONNECTION_MAX

(integer, >= 0) The maximum number of concurrent, inbound TCP connections to allow across all SMPP relay and server instantiations. A value of 0 (zero) indicates that there is no global limit on the number of connections. There may, however, be per relay or server limits imposed by a given relay or server instantiation. Default: 10,000

C.5.9 SMPP Relay Options

The SMS Gateway Server can have multiple instantiations of its SMPP relay, each with different characteristics chief of which will be the TCP port and interface listened on. Put differently, for each network interface and TCP port pair the SMPP relay listens on, distinct characteristics may be ascribed. These characteristics are specified using the options described in this section.

Each instantiation should be placed within an option group of the form:

```
[SMPP_RELAY=relay-name]
option-name-1=option-value-1
option-name-2=option-value-2
...
option-name-n=option-value-n
```

The string `relay-name` merely serves to differentiate this instantiation from other instantiations.

Table C-22 lists the SMPP relay configuration options.

Table C-22 SMPP Relay Options

Options	Default	Description
C.5.9.1 LISTEN_BACKLOG	255	Connection backlog for inbound SMPP client connections
LISTEN_CONNECTION_MAX	no default value	Maximum number of concurrent inbound connections
LISTEN_INTERFACE_ADDRESS	no default value	Network interface for inbound SMPP client connections
LISTEN_PORT	no default value	TCP port for inbound SMPP client connections
LISTEN_RECEIVE_TIMEOUT	600 s	Read timeout for inbound connections from SMPP clients
LISTEN_TRANSMIT_TIMEOUT	120 s	Write timeout for inbound connections from SMPP clients
MAKE_SOURCE_ADDRESSES_UNIQUE	1	Make relayed SMS source addresses unique and able to be replied to
SERVER_HOST	no default value	Host name or IP address of the SMPP server to relay to
SERVER_PORT	no default value	TCP port of the SMPP server to relay to
SERVER_RECEIVE_TIMEOUT	600 s	Read timeout for outbound SMPP server connections
SERVER_TRANSMIT_TIMEOUT	120 s	Write timeout for outbound SMPP server connections

C.5.9.1 LISTEN_BACKLOG

(*integer*, `in [0,255]`) Connection backlog allowed by the TCP stack for inbound SMPP client connections. The default value is 255.

LISTEN_CONNECTION_MAX

(*integer, >= 0*) The maximum number of concurrent, inbound TCP connections to allow for this SMPP relay instantiation. Note that this value will be ignored if it exceeds the global `LISTEN_CONNECTION_MAX` setting.

LISTEN_INTERFACE_ADDRESS

(*string,* `"INADDR_ANY"` *or dotted decimal IP address*) The IP address of the network interface to listen to for inbound SMPP client connections. May be either the string `"INADDR_ANY"` (all available interfaces) or an IP address in dotted decimal form. (For example, 193.168.100.1) The default value is `"INADDR_ANY"`. Clustered HA configurations will need to set this value to correspond to the HA logical IP address.

LISTEN_PORT

(integer, TCP port number) TCP port to bind to for accepting inbound SMPP client connections.

Specification of this option is mandatory; there is no default value for this option. Note also that there is no Internet Assigned Numbers Authority (IANA) assignment for this service.

LISTEN_RECEIVE_TIMEOUT

(integer, seconds > 0) Timeout to allow when waiting to read data from an SMPP client. The default value is 600 seconds (10 minutes).

LISTEN_TRANSMIT_TIMEOUT

(integer, seconds > 0) Timeout to allow when sending data to an SMPP client. The default value is 120 seconds (2 minutes).

MAKE_SOURCE_ADDRESSES_UNIQUE

(0 or 1) By default, the SMPP relay will append to each SMS source address a unique, ten digit string. The resulting SMS source address is then saved along with the other historical data. The result is a unique SMS address which may then be replied to by SMS users. The SMPP server will detect this address when used as an SMS destination address and will then send the SMS message to the correct email originator.

To disable this generating of unique SMS source addresses (for one-way SMS), specify a value of 0 (zero) for this option.

SERVER_HOST

(string, TCP hostname or dotted decimal IP address) SMPP server to relay SMPP client traffic to. Either a hostname or IP address may be specified. Specification of this option is mandatory; there is no default value for this option.

SERVER_PORT

(integer, TCP port number) TCP port for the remote SMPP server to which to relay. Specification of this option is mandatory; there is no default value for this option. There is no IANA assignment for this service; do not confuse with the IANA assignment for SNPP.

SERVER_RECEIVE_TIMEOUT

(integer, seconds > 0) Timeout to allow when waiting to read data from the SMPP server. The default value is 600 seconds (10 minutes).

SERVER_TRANSMIT_TIMEOUT

(integer, seconds > 0) Timeout to allow when sending data to the SMPP server. The default value is 120 seconds (2 minutes).

C.5.10 SMPP Server Options

The SMS Gateway Server can have multiple instantiations of its SMPP server, each with different characteristics chief of which will be the TCP port and interface listened on. Put differently, for each network interface and TCP port pair the SMPP server listens on, distinct characteristics may be ascribed. These characteristics are specified using the options described in this section.

Each instantiation should be placed within an option group of the form:

```
[SMPP_SERVER=server-name]
option-value-1=option-value-1
option-value-2=option-value-2
...
option-name-n=option-value-n
```

The string `server-name` merely serves to differentiate the instantiation from other instantiations.

Table C-23 lists the SMPP server configuration options.

Table C-23 SMPP Server Options

Options	Default	Description
C.5.10.1 LISTEN_BACKLOG	`255`	Connection backlog for inbound SMPP server connections
LISTEN_CONNECTION_MAX	no default value	Maximum number of concurrent inbound connections
LISTEN_INTERFACE_ADDRESS	no default value	Network interface for inbound SMPP server connections
LISTEN_PORT	no default value	TCP port for inbound SMPP server connections
LISTEN_RECEIVE_TIMEOUT	`600 s`	Read timeout for inbound SMPP server connections
LISTEN_TRANSMIT_TIMEOUT	`120 s`	Write timeout for inbound SMPP server connections

C.5.10.1 LISTEN_BACKLOG

(*integer in [0,255]*) Connection backlog allowed by the TCP stack for inbound SMPP client connections. The default value is 255.

LISTEN_CONNECTION_MAX

(integer >= 0) The maximum number of concurrent, inbound TCP connections to allow for this SMPP server instantiation. Note that this value will be ignored if it exceeds the global `LISTEN_CONNECTION_MAX` setting.

LISTEN_INTERFACE_ADDRESS

(string, `"INADDR_ANY"` *or dotted decimal IP address)* The IP address of the network interface to listen to for inbound SMPP client connections on. May be either the string `"INADDR_ANY"` (all available interfaces) or an IP address in dotted decimal form. (For example, 193.168.100.1.) The default value is `"INADDR_ANY"`.

LISTEN_PORT

(integer, TCP port number) TCP port to bind to for accepting inbound SMPP client connections. Specification of this option is mandatory; there is no default value for this option. Note that there is no IANA assignment for this service.

LISTEN_RECEIVE_TIMEOUT

(integer, seconds > 0) Timeout to allow when waiting to read data from an SMPP client. The default value is 600 seconds (10 minutes).

LISTEN_TRANSMIT_TIMEOUT

(integer, seconds > 0) Timeout to allow when sending data to an SMPP client. The default value is 120 seconds (2 minutes).

C.5.11 Gateway Profile Options

There may be zero or more gateway profiles. In the SMS Gateway Sever's configuration file, each gateway profile is declared within an option group in the following format:

```
[GATEWAY_PROFILE=profile-name]
option-name-1=option-value-1
option-name-2=option-value-2
...
option-name-n=option-value-n
```

The string `profile-name` merely serves to differentiate the profile from other origination profiles.

Table C-24 lists the SMS Gateway Server profile options.

Table C-24 SMS Gateway Server Profile Options

Options	Default	Description
C.5.11.1 CHANNEL	`sms`	Channel to enqueue message as
EMAIL_BODY_CHARSET	`US-ASCII`	Character set for email message bodies
EMAIL_HEADER_CHARSET	`US-ASCII`	Character set for email message headers
FROM_DOMAIN	no default value	Domain name for routing email back to SMS
PARSE_RE_0, PARSE_RE_1, ..., PARSE_RE_9	no default value	Regular expressions for parsing SMS message text
PROFILE	`GSM`	SMS profile to operate under: GSM, TDMA, or CDMA
SELECT_RE	no default value	Regular expression for selecting the plugin
SMSC_DEFAULT_CHARSET	`US-ASCII`	SMSC's default character set
USE_SMS_PRIORITY	`0`	Gateway SMS priority flags to email
USE_SMS_PRIVACY	`0`	Gateway SMS privacy indicators to email

C.5.11.1 CHANNEL

(string, 1-40 characters) Name of the MTA channel used to enqueue email messages. If not specified, then "sms" is assumed. The specified channel must be defined in the MTA's configuration.

EMAIL_BODY_CHARSET

(string, character set name) The character set to translate SMS text to prior to insertion into an email message's body. If necessary, the translated text will be MIME encoded. The default value is US-ASCII. If the SMS message contains glyphs not available in the charset, they will be converted to mnemonic characters, which may or may not be meaningful to the recipient.

A list of the character sets known to the MTA may be found in the following file:

```
installation-directory/config/charsets.txt
```

EMAIL_HEADER_CHARSET

(string, character set name) The character set to translate SMS text to prior to insertion into an RFC 822 `Subject: header` line. If necessary, the translated string will be MIME encoded. The default value is US-ASCII. If the SMS message contains glyphs not available in the charset, they will be converted to mnemonic characters, which may or may not be meaningful to the recipient

FROM_DOMAIN

(string, IP host name, 1-64 characters) Domain name to append to SMS source addresses when constructing envelope `From:` addresses for email messages. The name specified should be the correct name for routing email back to SMS. (For example, the host name associated with the MTA SMS channel.) If not specified, then the official host name of the channel specified with the `CHANNEL` option will be used.

PARSE_RE_0, PARSE_RE_1, ..., PARSE_RE_9

(string, UTF-8 regular expression) For mobile origination of email, the gateway profile needs to extract a destination email address from the text of the SMS message. This is done by means of one or more POSIX-compliant regular expressions (REs). The text of the SMS message will be evaluated by each regular expression until either a match producing a destination email address is found or the list of regular expressions exhausted.

> **Note -**
> Use of PARSE_RE_* and ROUTE_TO options are mutually exclusive. Use of both in the same gateway profile is a configuration error.

Each regular expression must be POSIX compliant and encoded in the UTF-8 character set. The regular expressions must output as string 0 the destination address. They may optionally output text to use in a `Subject:` header line as string 1, and text to use in the message body as string 2. Any text not "consumed" by the regular expression will also be used in the message body, following any text output as string 2.

The regular expressions will be tried in the order PARSE_RE_0, PARSE_RE_1, ..., up to PARSE_RE_9. If no regular expressions are specified, then the following default regular expression is used:

```
[ \t]*([^\( ]*)[ \t]*(?:\(([^\)]*)\))?[ \t]*(.*)
```

This default regular expression breaks into the following components:

```
[ \t]*
```

Ignore leading white space characters (SPACE and TAB).

```
([^\( ]*)
```

Destination email address. This is the first reported string.

```
[ \t]*
```

Ignore white space characters.

```
(?:\(([^\)]*)\))?
```

Optional subject text enclosed in parentheses. This is the second reported string. The leading `?:` causes the outer parentheses to not report a string. They are being used merely for grouping their contents together into a single RE for the trailing `?`. The trailing `?` causes this RE component to match only zero or one time and is equivalent to the expression:

```
{0,1}
```

{{[\t]*}}

Ignore white space characters.

`(.*)`

Remaining text to message body. This is the third reported string.

For example, with the above regular expression, the sample SMS message:

`dan@sesta.com(Testing)This is a test`

yields the email message:

```
To: dan@sesta.com
Subject: Testing
```

`This is a test`

As a second example, the SMS message:

`sue@sesta.com This is another test`

would yield:

`To: sue@sesta.com`

`This is another test`

Note that the SMS message, prior to evaluation with these regular expressions, will be translated to the UTF-16 encoding of Unicode. The translated text is then evaluated with the regular expressions which were previously converted from UTF-8 to UTF-16. The results of the evaluation are then translated to US-ASCII for the destination email address, `EMAIL_HEADER_CHARSET` for the `Subject:` text, if any, and `EMAIL_BODY_CHARSET` for the message body, if any.

PROFILE

(string, "GSM", "TDMA", or "CDMA") SMS profile to assume. Presently this information is only used to map SMS priority flags to RFC 822 `Priority:` header lines. Consequently, this option has no effect when `USE_SMS_PRIORITY=0` which is the default setting for that option.

SELECT_RE

(string, US-ASCII regular expression) A US-ASCII POSIX-compliant regular expression to compare against each SMS message's SMS destination address. If an SMS message's destination address matches this RE, then the SMS message will be sent through the gateway to email in accord with this gateway profile.

Note that since an SMS message's destination address is specified in the US-ASCII character set, this regular expression must also be expressed in US-ASCII.

SMSC_DEFAULT_CHARSET

(string, character set name) The name of the default character set used by the remote SMSC. The two common choices for this option are US-ASCII and UTF-16-BE (USC2). If not specified, US-ASCII is assumed.

USE_SMS_PRIORITY

(integer, 0 or 1) By default (with `USE_SMS_PRIORITY=0`), priority flags in SMS messages are ignored and not sent with the email messages. To have the priority flags passed with the email, specify `USE_SMS_PRIORITY=1`. When passed with the email, the mapping from SMS to email is as shown in Table C-25:

Table C-25 Priority Flag Mapping from SMS to Email

SMS Profile	SMS Priority Flag	Email Priority: Header Line
GSM	`0` (Non-priority)`1`, 2, 3 (Priority)	No header line (implies `Normal`)`Urgent`
TDMA	`0` (Bulk)`1` (Normal)2 (Urgent)3 (Very Urgent)	`Nonurgent}}No header line (implies {{Normal)UrgentUrgent`
CDMA	`0` (Normal)`1` (Interactive)2 (Urgent) 3 (Emergency)	No header line (implies `Normal)UrgentUrgent Urgent`

Note that the email `Priority:` header line values are `Nonurgent`, `Normal`, and `Urgent`.

USE_SMS_PRIVACY

(integer, 0 or 1) By default (with `USE_SMS_PRIVACY=0`), SMS privacy indications are ignored and not sent with the email messages. To have this information passed with the email, specify `USE_SMS_PRIVACY=1`. When passed along with email, the mapping from SMS to email is as shown in Table C-26:

Table C-26 Privacy Flags Mapping from SMS to Email

SMS Privacy Flag	Email Sensitivity: Header Line
0 (Not restricted)	No header line
1 (Restricted)	Personal
2 (Confidential)	Private
3 (Secret)	Company-confidential

Note that the values of the email Sensitivity: header line are Personal, Private, and Company-confidential.

C.5.12 Configuration Example for Two-Way SMS

Assumptions on Behavior

For the sake of this example, let us assume that the following behavior is desired:

- Email messages addressed to
  sms-id@sms.domain.com
  are to be sent to the SMS address
  sms-id
  and given a unique SMS source address in the range 000nnnnnnnnn.
- Mobile SMS messages addressed to the SMS address 000 are to be sent through the gateway to email with the email address extracted from the start of the SMS message text.
  For example, if the SMS message text is:
  jdoe@domain.com Interested in a movie?
  then the message "Interested in a movie?" is to be sent to jdoe@domain.com.
- SMS notifications sent to 000nnnnnnnnn are to be sent through the gateway to email and directed to the originator of the message being receipted.

In order to bring about this behavior, the following assumptions and assignments are made

Further Assumptions and Assignments

- The MTA's SMS channel uses the domain name sms.domain.com.
- The SMS Gateway Server runs on the host gateway.domain.com and uses:
  - TCP port 503 for its SMPP relay
  - TCP port 504 for its SMPP server
- The remote SMSC's SMPP server runs on the host smpp.domain.com and listens on TCP port 377.
- The remote SMSC's default character set is UCS2 (aka, UTF-16).

SMS Channel Configuration

To effect the above behavior, the following SMS channel configuration may be used in the imta.cnf file (add these lines to the bottom of the file):

```
(blank line)
sms
sms.domain.com
```

SMS Channel Option File

The channel's option file, sms_option, would then contain the following settings:

```
SMPP_SERVER=gateway.domain.com
SMPP_PORT=503
USE_HEADER_FROM=0
DEFAULT_SOURCE_ADDRESS=000
GATEWAY_PROFILE=sms1
SMSC_DEFAULT_CHARSET=UCS2
```

SMS Gateway Server Configuration

Finally, the Gateway Server configuration file, `sms_gateway.cnf`, should look like the following:

```
HISTORY_FILE_DIRECTORY=/sms_gateway_cache/
[SMPP_RELAY=relay1]
LISTEN_PORT=503SERVER_HOST=smpp.domain.com
SERVER_PORT=377

[SMPP_SERVER=server1]
LISTEN_PORT=504

[GATEWAY_PROFILE=sms1]
SELECT_RE=000([0-9]{10,10}){0,1}
SMSC_DEFAULT_CHARSET=UCS2
```

Testing This Configuration

If you do not have an SMSC to test on, you may want to perform some loopback tests. With some additional settings in the `sms_option` file, some simple loop back tests may be performed for the above configuration.

C.5.12.1 Additional sms_option File Settings

The additional settings for the `sms_option` file are:

```
! So that we don&rsquo;t add text to the body of the SMS message
FROM_FORMAT=
SUBJECT_FORMAT=
CONTENT_PREFIX=
```

Without these settings, an email containing:

```
user@domain.com (Sample subject) Sample text
```

would get converted into the SMS message:

```
From:user@domain.com Subject:Sample Subject Msg:Sample text
```

That, in turn, would not be in the format expected by the mobile-to-email code, which wants to see:

```
user@domain.com (Sample subject) Sample text
```

Hence the need (for loopback testing) to specify empty strings for the `FROM_FORMAT`, `SUBJECT_FORMAT`, and `CONTENT_PREFIX` options.

Performing the Loopback Test

Send test email messages addressed to `000@sms.domain.com`, such as:

```
user@domain.com (Test message) This is a test message which should loop back
```

The result is that this email message should be routed back to the email recipient `user@domain.com`. Be sure to have added `sms.domain.com` to your DNS or host tables for the test.

C.6 SMS Gateway Server Storage Requirements

To determine the amount of resources you will need for the SMS Gateway Server, use the numbers you generate from the requirements in Table C-27 along with your expected number of relayed messages per second and the `RECORD_LIFETIME` setting.

Table C-27 covers the requirements for the historical records, the SMPP relay, and SMPP server.

Table C-27 SMS Gateway Server Storage Requirements

Component	Requirements

In-memory historical record	Each relayed message requires 33+m+s bytes of virtual memory, where m is the length of the message's SMS message ID (1 <= m <=64) and s is the length of the message's SMS source address (1 <=s <= 20).
	
	When MAKE_SOURCE_ADDRESS_UNIQUE=0, then only 16+m bytes are used. For 64-bit operating systems, 49+m+s bytes of virtual memory are consumed per record [24+m when MAKE_SOURCE_ADDRESS_UNIQUE=0].
	
	Note also, that the heap allocator may actually allocate larger size pieces of virtual memory for each record.
	
	The maximum number of records is 43 billion:
	
	`(2**32-1)`
	
	For fewer than 16.8 million records:
	
	`(2**24)`
	
	the hash table consumes approximately 16 Mb.
	For fewer than 67.1 million records:
	
	`(2**26)`
	
	the hash table consumes approximately 64 Mb; for more than 67.1 million records, the hash table consumes approximately 256 Mb.
	
	Double the memory consumptions for 64 bit operating systems.
	
	These consumptions are in addition to the memory consumption required for each record itself.

On-disk historical record	Each relayed message requires on average the following number of bytes:
	
	`81+m+2s+3a+S+2i`
	
	where:
	
	• m is the average length of SMS message IDs, and 1<=m<=64
	• s is the average length of SMS source addresses, and 1 <= s <= 20
	• a is the average length of email addresses, and 3 <= a <= 129
	• S is the average length of Subject: header lines, and 0 <= S<= 80
	
	`78+m+3a+S+2i`

SMPP relay	Each relayed SMPP session consumes two TCP sockets: one with the local SMPP client and another with the remote SMPP server. Approximately 1 Kb of virtual memory is consumed per connection on 32 bit operating systems; 2 Kb on 64 bit operating systems.
SMPP server	Each incoming connection consumes a TCP socket. Approximately 1 Kb of virtual memory is consumed per connection on 32 bit operating systems; 2 Kb on 64 bit operating systems.

For instance, if on average 50 messages per second are expected to be relayed, SMS source addresses are 13 bytes long, SMS message IDs have a typical length of 12 bytes, email addresses 24 bytes, `Subject:` lines 40 bytes, email message and envelope IDs 40 bytes each, and historical data is retained for 7 days, then:

- There will be 30.24 million historical records to store, each on average 58 bytes in memory and 311 bytes long on disk;
- The in-memory consumption of the historical records will be about 1.70 Gb (1.63 Gb + 64 Mb); and
- The on-disk storage will be approximately 8.76 Gb.

While a sufficiency of disk may be supplied to handle any on disk requirements, the virtual memory requirement on a 32-bit machine will be a hard limit of approximately 2 Gb. To reduce the amount of virtual memory or disk storage required, use the `RECORD_LIFETIME` option to reduce the length of time records are retained.

C.7 SMS Configuration Examples

The following examples outline a couple of ways to configure one-way and two-way SMS:

- Configuring Messaging Server for One-Way SMS
- Configuring Messaging Server for Two-Way SMS

Configuring Messaging Server for One-Way SMS

Configuring Messaging Server for One-Way SMS

The following example describes one method for configuring one-way SMS in Messaging Server.

1. Log in as either root or the Messaging Server user. Messaging Server configuration files must be owned by the Messaging Server user. For information on creating a Messaging Server user, see Creating UNIX System Users and Groups.

2. Define a rewrite rule for messages sent to the SMS channel.
   In the upper section of the `imta.cnf` file, enter a rewrite rule in the following format:
   `smpp-host-name $U@smpp-host-name`
   For example:
   `sms $U@sms-handle`
   The format in this example is not a requirement; see Adding the Channel Definition and Rewrite Rules for some alternatives.

3. Define an MTA Channel for SMS.
   The destination of the channel must be a Short Message Peer to Peer (SMPP) server that, in turn, communicates with an external short-message service center (SMSC). For communication with multiple SMSCs, you need to configure a separate channel for each SMSC.

   To define an MTA Channel for SMS:
   a. Choose a name for the channel.
      The name must either be *sms* or a name of the form *smsname*, for example, sms_ch1.

   b. Add a channel definition at the bottom of the `imta.cnf` file, which is in the `/opt/sun/comms/messaging64/config/` directory. The definition must be preceded by a blank line.

      The following example defines a channel, `sms`, with the SMPP server `sms-handle` as its endpoint. The `notificationchannel` option in the example is required for SMS (see the step for configuring the process channel. It specifies a process channel for handling delivery status notifications (DSNs) generated by the SMS channel. The process channel's name must have the following format: `process_`*`channelName`*.

```
[Blank line]
sms port 4321 notificationchannel process_sms threaddepth 20
backoff "pt2m" "pt5m" "pt10m" "pt30m" notices 1
sms-handle
```

4. Configure the process channel for the SMS channel.
   Below the definition of the SMS channel, add a process channel with the name that you assigned to the `notificationchannel` channel option, earlier, and assign it a source filter, as follows:

```
[Blank line]
process_sms sourcefilter
file:/opt/sun/comms/messaging64/config/process_sms.filter
process_sms-daemon
```

5. Save the `imta.cnf` file.
6. Create a filter definition file for the source filter, named as in the preceding step, and enter a definition as in the following example:

```
require ["body","imap4flags"];
if body :raw :contains "Action: failed"
{
addflag "sms";
addflag "smserror";
}
```

7. Save the filter definition file.
8. Since the filter definition includes a Sieve body test, make sure that the MTA option file, `option.dat`, contains: `ENABLE_SIEVE_BODY=1`.
9. (Optional) Create a channel option file. See Creating an SMS Channel Option File for more information.

> **Note**
> The SMS channel option file contains site-specific parameters required for the operation of the channel. Contact the SMS service provider to get the info needed to add to the channel option file. A channel option file is not required for SMS.

10. If running a compiled configuration, recompile the MTA configuration to capture the changes you have made:

```
./imsimta cnbuild
```

11. Restart Messaging Server.

```
./stop-msg
./start-msg
```

Configuring Messaging Server for Two-Way SMS

Configuring Messaging Server for Two-Way SMS (Legacy Configuration)

The following example describes one method for configuring two-way SMS in Messaging Server:

1. Log in as either root or the Messaging Server user. Messaging Server configuration files must be owned by the Messaging Server user. For information on creating a Messaging Server user, see Creating UNIX System Users and Groups.
2. Define a rewrite rule for messages sent to the SMS channel.
   In the upper section of the `imta.cnf` file, enter a rewrite rule in the following format:
   `smpp-host-name $U@smpp-host-name`
   For example,
   `sms $U@sms-handle`
   For alternative examples on formatting channel definitions, see Adding the Channel Definition and Rewrite Rules.
3. Define an MTA Channel for SMS.
   The destination of the channel must be a Short Message Peer to Peer (SMPP) server that, in turn, communicates with an external short-message service center (SMSC). For communication with multiple SMSCs, you need to configure a separate channel for each SMSC.
   a. Choose a name for the channel.
      The name must either be *sms* or a name of the form *smsname*, for example, sms_ch1.

   b. Add a channel definition at the bottom of the `imta.cnf` file, which is in the `/opt/sun/comms/messaging64/config/` directory. The definition must be preceded by a blank line.

      The definition must contain the following elements:

Channel name	The channel name must be the first element in the channel definition. The name must either be *sms* or a name of the form *sms_name*, for example, sms_ch1.
Channel destination	The destination must be a Short Message Peer to Peer (SMPP) server that, in turn, communicates with an external short-message service center (SMSC). For communication with multiple SMSCs, you need to configure a separate channel for each SMSC (see Adding Additional SMS Channels). The Destination host must be the last element in the channel definition.
port	The TCP port the SMPP server listens on.
Source filter	A `sourcefilter` option that specifies the location of a source filter for SMS: `file:/opt/sun/comms/messaging64/config/sms.filter`
Notification channel option	A `notificationchannel` option that specifies a process channel for handling delivery status notifications (DSNs) generated by the SMS channel. The process channel's name must have the following format: `process_`*`channelName`*. For more information, see notificationchannel Channel Option.

   c. Determine the value of required elements and any additional options you choose.
   d. Add a channel definition at the bottom of the `imta.cnf` file, which is in the `/opt/sun/comms/messaging64/config/` directory. The definition must be preceded

by a blank line.
The following example defines a channel, `sms` with the SMPP server `sms-handle` as its endpoint.

```
[Blank line]
sms port 4321 sourcefilter
file:/opt/sun/comms/messaging64/config/sms.filter
notificationchannel process_sms threaddepth 20 backoff "pt2m"
"pt5m" "pt10m" "pt30m" notices 1
sms-handle
```

4. Create the SMS channel filter to file the incoming SMS messages into a given folder and flag them. The flag will be used by Convergence for notifications:

```
bash-3.00# more /opt/sun/comms/messaging64/config/sms.filter
require ["fileinto", "imap4flags"];
#if header :contains "Message-context" "pager-message"
#{
# addflag "sms";
# fileinto "SMS";
#}
fileinto :flags "sms" "SMS";
```

5. Define a process channel for the SMS channel.
   Below the definition of the SMS channel, add a process channel with the name that you assigned to the `notificationchannel` option and assign it a source filter, as follows:

```
[Blank line]
process_sms sourcefilter
file:/opt/sun/comms/messaging64/config/process_sms.filter
process_sms-daemon
```

6. Save the `imta.cnf` file.
7. Create a filter definition file for the source filter and enter a definition, as in the following example. This is the source filter for the `process_sms` channel:

```
require ["body","imap4flags"];
if body :raw :contains "Action: failed"
{
addflag "sms";
addflag "smserror";
}
```

8. Save the process definition file.
9. Since the process definition includes a Sieve body test, make sure that the MTA option file, `option.dat`, contains: `ENABLE_SIEVE_BODY=1`.
10. (Optional) Create an channel option file. See Creating an SMS Channel Option File for more information.

> **Note**
> The SMS channel option file contains site-specific parameters required for the operation of the channel. A channel option file is not required for SMS.

11. Enable JMQ notification parameters in Messaging Server (see Configuring a JMQ Notification Service. To enable the JMQ Notification plug-in, the `jmqnotify.NewMsg.enable` parameter must be set to 1. The notification parameters are configured for JMQ to create non-delivery failure notifications of any SMS messages.
    A sample notification configuration follows. Note that the parameter `jmqnotify.DestinationType` can be set to either `queue` or `topic`. In the example, it is set to `queue`.

```
./configutil -o local.store.notifyplugin.jmqnotify.NewMsg.enable -v 1
./configutil -o local.store.notifyplugin.jmqnotify.jmqHost -v
"127.0.0.1"
./configutil -o local.store.notifyplugin.jmqnotify.jmqPort -v "7777"
./configutil -o local.store.notifyplugin.jmqnotify.jmqUser -v "user1"
./configutil -o local.store.notifyplugin.jmqnotify.jmqpwd -v "xxx"
./configutil -o local.store.notifyplugin.jmqnotify.DestinationType -v
"queue"
./configutil -o local.store.notifyplugin.jmqnotify.jmqQueue -v "ucsms1"

./configutil -o local.store.notifyplugin.jmqnotify.Priority -v 3
./configutil -o local.store.notifyplugin.jmqnotify.ttl -v 1000
./configutil -o local.store.notifyplugin.jmqnotify.Persistent -v 1
./configutil -o local.store.notifyplugin -v
'libibiff$ms-internal$libjmqnotify$jmqnotify'
./configutil -o local.store.notifyplugin.jmqnotify.maxheadersize -v
1024
./configutil -o local.store.notifyplugin.jmqnotify.noneinbox.enable -v
1
./configutil -o local.store.notifyplugin.jmqnotify.msgflags.enable -v 0

./configutil -o local.store.notifyplugin.jmqnotify.readmsg.enable -v 0
```

12. Enable the SMS gateway server; set the `configutil` parameter `local.smsgateway.enable` to 1:

```
configutil -o local.smsgateway.enable -v 1
```

13. Set SMS-gateway configuration options in the `sms_gateway.cnf` file. The file is in `/opt/sun/comms/messaging64/config/`.
    The following table lists typical SMS gateway configuration options. For additional options, see C.5.7 Configuration Options. Note that you must set the `TEXT_TO_SUBJECT` option to 1.

[GATEWAY_PROFILE=Gateway]	Name of the gateway profile.
TEXT_TO_SUBJECT=1	Convert incoming SMS message body into e-mail subject.
CHANNEL=sms	Name of the SMS channel given in imta.cnf.
SELECT_RE=([0-9]*)	If an SMS message's destination address matches this RE, the message is sent through the gateway to email in accord with this gateway profile. The RE in the example will match any number.
SMSC_DEFAULT_CHARSET=UTF-8	Default character set to be used by the SMS gateway.
EMAIL_BODY_CHARSET=UTF-8	The character set to translate SMS text into prior to insertion into an email message's body.
[SMPP_RELAY=Relay]	Any name may be used for the relay's name. All that matters is that the name not be used for any other SMPP relay instantiation within the same configuration file.
LISTEN_PORT=8500	The port on which the Relay listens.
SERVER_HOST=127.0.0.1	The remote SMSC server's IP address.
SERVER_PORT=950	The port the remote SMSC server listens on.
[SMPP_SERVER=Server]	The local SMPP server configuration.
LISTEN_PORT=4080	The port the local SMPP server listens on.

14. If running a compiled configuration, recompile the Messaging Server configuration to capture the changes you have made:

```
./imsimta cnbuild
```

15. Restart Messaging Server:

```
./stop-msg
./start-msg
```

Chapter 35. Administering Messaging Server Remotely

Administering Oracle Communications Messaging Server Remotely

> Starting with **Messaging Server 7 Update 4**, this command is considered deprecated. There are no plans to enhance this command and it might be removed in a future release.
>
> The `msgssh` command provides inferior security when compared to a regular `ssh` session with appropriately configured RBAC (Solaris OS) or `sudo` (Red Hat Linux). The latter provides a more flexible framework for remote administration, because it supports administration of co-located products in addition to just Messaging Server.
>
> The `mgssh` command was introduced in **Messaging Server 7**. This command was renamed from `msgadm` to `msgssh` in **Messaging Server 7 Update 2** to better reflect its function.

The `msgssh` command provides an ssh-based remote administration shell for Messaging Server.

Topics:

Overview of Remotely Administering Messaging Server

Remote administration of Messaging Server consists of four parts:

- `msgssh`, a client utility to remotely connect to Messaging Server instances.
- `msgadmd`, the server part that runs on the Messaging Server instances.
- Additional administrative tools in the `bin` directory: `showconfig`, `showlog`, `writeconfig`.
- Additional command wrappers: `grep`, `help`, `less`, `ls`

Configuring Messaging Server for Remote Administration

To enable remote administration for a Messaging Server deployment, you define the following items:

- A set of Messaging Server systems you want to remotely manage (the "*Messaging Server instances*")
- A user name on the Messaging Server instances (the "*local admin user*") – probably the "*mailsrv*" userid
- A server from which you manage the Messaging Server systems (the "*remote management station*")
- A user name on the remote management station which performs the administration (the "*remote admin user*")

The same user name can be used on the remote management station and the Messaging Server instances, but for completeness, the following discussion separates the concepts.

> **Note**
> With the present version of this tool, the *local admin user* must be `root` for privileged commands, including `start-msg`, `stop-msg` and `configure`.

To Configure a Messaging Server Deployment for Remote Administration

1. Perform the following steps on the *remote management station*:
   a. Log in as the *remote admin user*.
   b. Create an SSH key by using the `ssh-keygen` command, if a key does not already exist. For more information, see the `ssh-keygen(1)` man page.
   c. Copy the `msgssh` command from a Messaging Server instance.
   The `msgssh` command is located in the *msg-svr-base*`/bin` directory. *msg-svr-base* refers to the directory where Messaging Server is installed. Starting with Messaging Server 7.0, the default is `/opt/sun/comms/messaging`.
2. Perform the following steps on each *Messaging Server instance*:
   a. If you are using the *mailsrv* userid, make sure the account has been created, has a home directory with appropriate ownership and permissions, and you can log in to that account.
   b. If you are using `root` for the *local admin user*, you need to enable log in remotely as `root` by setting `PermitRootLogin yes` in `/etc/ssh/sshd_config`.

      > **Note**
      > For security reasons, do not do this this for operational deployments unless additional security (such as restricted physical network access) protects the machine. Consult your security administrator to determine what is appropriate for your site's security policy.

   c. Copy the contents of the public key (for example, `id_rsa.pub`) from Step 1b for the *remote admin user* to the *local admin user's* `.ssh/authorized_keys` file, preceded by the phrase `'command="`*msg-svr-base*`/lib/msgadmd" '`.
   That is, the `.ssh/authorized_keys` file should contain a line similar to the following line:
   `command="`*msg-svr-base*`/lib/msgadmd" ssh-rsa AAAAB3NzaC1yc2...`
   This permits `remoteadminuser@remotemgmthost` users to authenticate to `localadminuser@messagingserver` by using their SSH key but limits command execution to *msg-svr-base*`/lib/msgadmd`.

      > **Note**
      > Be sure that file permissions are set correctly. The `.ssh/authorized_keys` file must be mode `600`, and the `.ssh` directory and home directory must not be world writable.

You do not need to configure the Messaging Server instance at this stage. You can configure Messaging Server remotely by using this tool if the localuser is `root`.

msgssh Usage

The `msgssh` command has the following usage:

```
Usage: msgssh [OPTIONS]
       Run Messaging Server Admin shell on remote systems
Usage: msgssh [OPTIONS] <command>
       Run Messaging Server Admin command on remote systems

The accepted values for OPTIONS are:

-?, --help   Display this help list
-t, --target "[<user>@]<host> ..."
             Messaging Server target list (space-separated)
             If <user>@ is not specified, the default is used
             If no -t is specified, the default is 127.0.0.1
-u, --user   Specify default user, default is root
-q, --quiet  Quiet mode (don't display "Connecting to..." message)
```

Invoking a Remote Administration Shell

To run `msgssh` as an interactive shell, don't specify a command. Example:

```
remoteuser@remotemgmthost % msgssh --target root@msg1
Connecting to root@msg1...
Messaging Server Administrative Shell Commands
   "help [<command>]" for help; ^D or "exit" to logout
   <TAB> for command line completion

MoveUser              imexpire              migrate-config
UpgradeMsg5toMsg7.pl  iminitquota           mkbackupdir
config-mfwk           imkill                msgssh
config-servicetags    immonitor-access      msgcert
config-vcsha          imquotacheck          patch-config
configtoxml           ims_db_upgrade        readership
configure             imsbackup             reconstruct
configutil            imsconnutil           refresh
counterutil           imscripter            relinker
deliver               imsexport             setconf
getconf               imsimport             showconfig
grep                  imsimta               showlog
ha_ip_config          imsrestore            spfquery
hashdir               init-config           start-msg
help                  install-newconfig     stop-msg
imarchive             less                  uninstall-newconfig
imcheck               ls                    useconfig
imdbverify            mboxutil              writeconfig
msgssh >
```

You can now enter commands at the "`msgssh >`" prompt with command-line completion and command history, e.g.,:

```
msgssh > configutil -o local.imta.enable
1
```

Invoking a Remote Administration Command

`msgssh` can be invoked to execute the same command on multiple machines. Example:

```
remoteuser@remotemgmthost % msgssh --target "msg1 msg2 msg3" configutil
-o foo -v bar
Connecting to root@msg1...
OK SET
WARNING: unrecognized option foo has been set
Connection to msg1 closed.
Connecting to root@msg2...
OK SET
WARNING: unrecognized option foo has been set
Connection to msg2 closed.
Connecting to root@msg3...
OK SET
WARNING: unrecognized option foo has been set
Connection to msg3 closed.
```

Shell conventions such as separating multiple commands with ';' is also possible.

Supporting CLIs for Remote Administration

The default path is limited to commands in *msg-svr-base*`/bin`, *msg-svr-base*`/lib/msgssh` and shell builtins. These limits are not privilege based, so it should be assumed the shell has unrestricted access to the capabilities of the admin user even if it appears otherwise.

> To facilitate common tasks under remote administration, the following commands were added in **Messaging Server 7.0**.

```
grep
        Search command output for a pattern
        Usage: <command> | grep [options] <pattern>
help
        Display help for command usage
        Usage: help [<command> | --help | -?]
less
        Display output of a command or a file one screen at a time
        Usage: [<command> "|"] less [OPTIONS] [ <file> | --help | -? ]
        For more detailed help run: "less --morehelp"
ls
        List Messaging Server Administrative Shell Commands
        Usage: ls [--help | -?]
showconfig
        Display config file listing or display config file(s)
        Usage: showconfig [ --help | -? ] [ <file> ... ]
        Display filtered results
        Usage: showconfig [ --help | -? ] [ <file> ... ] | grep
<pattern>
showlog
        Display log file listing or display log file(s)
        Usage: showlog [ --help | -? ] [ <file> ... ]
        Display filtered results
        Usage: showlog [ --help | -? ] [ <file> ... ] | grep <pattern>
writeconfig
        Write config file from standard input.
        Usage: writeconfig [ --help | -? ] <file>
```

Chapter 36. Configuring IMAP IDLE

Configuring IMAP IDLE

Use of ENS for IMAP IDLE notification is recommended and is the only supported option as of **Messaging Server 7 Update 4** and later. Use of JMQ for IMAP IDLE was briefly supported for Messaging Server 7 through 7 Update 3, and is not recommended.

If you have not updated to Messaging Server 7 Update 4, you can enable IMAP IDLE by updating to Messaging Server 7 Update 4 or later. Alternatively, you can follow the instructions in Configuring IMAP IDLE with ENS. For additional information about IMAP IDLE changes introduced with Messaging Server 7 Update 4, see Features Introduced in Messaging Server 7 Update 4.

Topics:

Benefits of Using IMAP IDLE

The IMAP IDLE extension to the IMAP specification, defined in RFC 2177, allows an IMAP server to notify the mail client when new messages arrive and other updates take place in a user's mailbox. The IMAP IDLE feature has the following benefits:

- Mail clients do not have to poll the IMAP server for incoming messages.
  Eliminating client polling reduces the workload on the IMAP server and enhances the server's performance. Client polling is most wasteful when a user receives few or no messages; the client continues to poll at the configured interval, typically every 5 or 10 minutes.

- A mail client displays a new message to the user much closer to the actual time it arrives in the user's mailbox. A change in message status is also displayed in near-realtime.
  The IMAP server does not have to wait for the next IMAP polling message before it can notify the client of a new or updated mail message. Instead, the IMAP server receives a notification as soon as a new message arrives or a message changes status. The server then notifies the client through the IMAP protocol.

Configuring IMAP IDLE with ENS

Starting with Messaging Server 7 Update 4, IMAP IDLE is enabled by default and it uses ENS. If you have a previous version of Messaging Server, use these instructions to configure IMAP IDLE with ENS.

In Messaging Server 7 Update 4, IMAP IDLE with ENS has the following default behavior:

- The initial configuration sets the `local.ens.enable` parameter to 1. If you upgraded to Messaging Server 7 Update 4, verify this is set to make sure IMAP IDLE is enabled.
- Every message store has its own `enpd` server.
- The `imapd` process, store delivery channels, and store utilities report changes to the `enpd` server

on the local store.

- Some additional configuration is helpful for improved security, HA, and flag updates, as explained in To Configure IMAP IDLE with ENS.
- IMAP IDLE does not require that events be aggregated to a single `enpd` server and the IDLE event distribution is more efficient if each store uses its own `enpd` server.

Prerequisites for Configuring IMAP IDLE with ENS

To enable ENS, set `local.ens.enable` to 1.

To use IMAP IDLE with ENS, you must configure the following ENS components:

- An `enpd` server on at least one host
- The `iBiff` notification plug-in on all message store hosts

For information on configuring ENS for Messaging Server, see *Unified Communications Suite Event Notification Service Guide*.

For information on configuring the `iBiff` notification plug-in, see Loading the ENS Publisher in Messaging Server.

To Configure IMAP IDLE with ENS

> The examples in this task use the default value of the Messaging Server 7 installation path, `/opt/sun/comms/messaging64`. Where applicable, differences between Messaging Server 7 Update 4, and Messaging Server 7 Update 3 and prior releases, are noted.

1. Configure the `enpd` server to accept connections only from the hosts running the message stores.
   - Messaging Server 7 Update 4: Allow or restrict connections by configuring the `service.ens.domainallowed` and `service.ens.domainnotallowed` parameters as necessary.
     These options work the same way as the equivalent options for POP, IMAP, and HTTP.
     These options replace the functionality of the ENS_ACCESS environment variable that was included in the legacy ENS server.
   - Messaging Server 7 Update 3 and prior releases: To restrict connections to message-store hosts, set the ENS_ACCESS environment variable. The environment variable sets a list of permissions allowing access to enpd. The syntax is as follows:

     ```
     setenv ENS_ACCESS 'allowdeny ipaddress|mask;
     allowdeny ipaddress|mask; ...'
     ```

     where
     `allowdeny` can be either {+} (to specify allow) or `---` (to specify deny)
     `ipaddress` specifies a dotted-decimal IP address
     `mask` specifies a dotted-decimal IP address mask
     Examples:
     The following example allows access to the local host only:

     ```
     setenv ENS_ACCESS '+127.0.0.1|255.255.255.255'
     ```

     The following example allows access to the local host and all IP addresses `192.168.0.*` except `192.168.0.17`:

```
setenv ENS_ACCESS
'+192.168.0.1|255.255.255.0;+127.0.0.1|255.255.255.255; \
-192.168.0.17;255.255.255.255'
```

2. Run the `configutil` utility to specify the name of the host where the ENS server is running.
   - Messaging Server 7 Update 4:
     For HA and multi-install systems, you still need to follow this step:

     ```
     cd /opt/sun/comms/messaging64
     ./configutil -o local.store.notifyplugin.ms-internal.enshost
     -v "ipaddress"
     ```

     where **`ipaddress`** specifies a dotted-decimal IP address of the ENS host machine.
     Example:

     ```
     cd /opt/sun/comms/messaging64
     ./configutil -o local.store.notifyplugin.ms-internal.enshost
     -v "127.0.0.1"
     ```

   - Messaging Server 7 Update 3 and prior releases:

     ```
     cd /opt/sun/comms/messaging64
     ./configutil -o local.store.notifyplugin.enshost -v
     "ipaddress"
     ```

     where **`ipaddress`** specifies a dotted-decimal IP address of the ENS host machine.
     Example:

     ```
     cd /opt/sun/comms/messaging64
     ./configutil -o local.store.notifyplugin.enshost -v
     "127.0.0.1"
     ```

3. Messaging Server 7 Update 3 and prior releases only: Specify the event key to use for notifications.
   If the ENS event key (`ensEventKey`) is set to its default value, IMAP IDLE does not operate. You must configure the `ensEventKey` value to end with `%M`. The string `%M` is a substitution code that is replaced by the name of the mailbox in which the event occurred.
   Run the following `configutil` command:

   ```
   ./configutil -o local.store.notifyplugin.enseventkey -v "eventkey"
   ```

   where `eventkey` is a unique identifier used by ENS. Its default value is `enp://127.0.0.1/store`. The host-name portion of the event key is not used to determine the host where ENS is running; it is simply part of the identifier.
   Example:

   ```
   ./configutil -o local.store.notifyplugin.enseventkey -v
   "enp://127.0.0.1/store/%M"
   ```

4. Messaging Server 7 Update 3 and prior releases only: Load the `libibiff` notification plug-in file, which enables the ENS publisher for Messaging Server. (In Messaging Server 7 Update 4 and later, iBiff is loaded by default.)
   Run the following `configutil` command (this is for 7.0p3 or later):

   ```
   ./configutil -o local.store.notifyplugin -v "lib/libibiff"
   ```

   Or you can use the full path (for versions prior to 7.0p3):

   ```
   ./configutil -o local.store.notifyplugin -v
   "/opt/sun/comms/messaging/lib/libibiff"
   ```

5. Messaging Server 7 Update 3 and prior releases only: Enable notifications to be sent from all user mailboxes, not only the inbox.
   By default, notifications are generated only by events that occur in the inbox. However, the IMAP IDLE RFC (2177) specifies that IDLE must notify clients whenever an event occurs in any mailbox.

   To comply with the RFC, the IMAP IDLE feature requires that notifications be enabled for all mailboxes. If they are not, the IMAP server will not advertise the IDLE capability.
   To configure notifications for all mailboxes, set the `configutil` command, `noneinbox`, to a value of 1:

   ```
   ./configutil -o local.store.notifyplugin.noneinbox.enable -v 1
   ```

   where `-v 1` enables notifications from all mailboxes.

6. Configure immediate flag update:

   ```
   configutil -o local.imap.immediateflagupdate -v 1
   (was "0")
   ```

7. Stop, then restart Messaging Server.

   ```
   cd /opt/sun/comms/messaging/sbin
   ./stop-msg
   ./start-msg
   ```

8. Verify that the IMAP services now include the IDLE feature. Use telnet to connect to the IMAP host and port.

   ```
   telnet IMAP_hostname port
   ```

   Example:

```
telnet myhost imap
trying 192.18.01.44 ...
connected to myhost.siroe.com
* OK [CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE
UIDPLUS
CHILDREN BINARY UNSELECT SORT LANGUAGE STARTTLS IDLE XSENDER
X-NETSCAPE
XSERVERINFO X-SUN-SORT X-SUN-IMAP X-ANNOTATEMORE AUTH=PLAIN]
myhost.siroe.com IMAP4 service (Sun Java(tm) System
Messaging Server 7.3-11.01 (built Sep 9 2009))
```

To Disable IMAP IDLE

To disable IMAP IDLE, set `service.IMAP.capability.idle` to 0 (default is 1).

Configuring IMAP IDLE with JMQ

> **Note**
> IMAP IDLE with JMQ is not recommended and ENS must be used as of Messaging Server 7 Update 4.

Using the JMQ notification service to produce IMAP IDLE notifications was a feature introduced in **Messaging Server 7**. Starting with **Messaging Server 7 Update 4**, IMAP IDLE is enabled by default and it uses ENS.

> The examples in this task use the default value of the Messaging Server 7 installation path, `/opt/sun/comms/messaging64`.

1. Modify the JMQ configuration file `/etc/imq/imqbrokerd.conf`.

```
replace:

AUTOSTART=NO

with:

AUTOSTART=YES
```

2. Start Java Message Queue.

```
/etc/init.d/imq start
```

3. Enable the IMAP IDLE parameter.
   Run the following `configutil` command:

```
cd /opt/sun/comms/messaging64/sbin
configutil -o service.imap.idle -v 1
(was "0")
```

4. Enable immediate flag update.

```
configutil -o local.imap.immediateflagupdate -v 1
(was "0")
```

5. Start Messaging Server.

```
start-msg
```

6. Verify that IMAP IDLE works. Run an IMAP IDLE session like the following example.

```
,----
| (112 root) telnet localhost 143
| Trying 127.0.0.1...
| Connected to localhost.
| Escape character is '^]'.
| * OK CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE
UIDPLUS CHILDREN BINARY UNSELECT SORT CATENATE URLAUTH LANGUAGE
ESEARCH ESORT THREAD=ORDEREDSUBJECT THREAD=REFERENCES ENABLE
CONTEXT=SEARCH CONTEXT=SORT WITHIN SASL-IR XSENDER X-NETSCAPE
XSERVERINFO X-SUN-SORT ANNOTATE-EXPERIMENT-1 X-UNAUTHENTICATE
X-SUN-IMAP X-ANNOTATEMORE XUM1 IDLE XREFRESH AUTH=PLAIN
worfin-1.red.iplanet.com IMAP4 service (Sun Java(tm) System
Messaging Server 7.0-0.04 32bit (built May 8 2008))
| a login <admin> <password>
| a OK User logged in
| b select inbox
| * FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
| * OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen
\*)]
| * 0 EXISTS
| * 0 RECENT
| * OK [UIDVALIDITY 1213216485]
| * OK [UIDNEXT 1]
| b OK [READ-WRITE] Completed
| c idle
| + idling
`----
```

7. Next, send an email in a separate window. You get the notification in the telnet window:

```
,----
| * 1 EXISTS
| * 1 RECENT
`----
```

More Information on JMQ and Messaging Server

For information about using JMQ to produce notifications with Messaging Server, see JMQ Notification.

For instructions on how to enable JMQ, see Enabling JMQ Notification (Example).

Chapter 37. BURL Support for SMTP SUBMIT

Enabling BURL Support for SMTP SUBMIT

Starting with Messaging Server 7, the MTA supports the BURL extension to SMTP SUBMIT, defined in RFC 4468 (Message Submission BURL Extension), and the Message Store IMAP server supports RFC 4467 (IMAP - URLAUTH Extension). BURL permits an email client to refer, in submitted messages, to content to be retrieved (using the IMAP URLAUTH extension) from an IMAP server. This allows email clients to submit messages including content without having to first download that content to the client and then upload (submit it) directly to the SMTP SUBMIT server itself: to include content by supplying a URL reference (including an authentication token allowing access) to the location of the content on an IMAP server. Availability of the BURL extension is controlled by the `BURL_ACCESS` mapping table, discussed below, and the MTA options `IMAP_USERNAME` and `IMAP_PASSWORD`.

For the BURL extension to be made available, a `BURL_ACCESS` mapping table must be defined. Note that BURL is specifically an SMTP SUBMIT feature; in terms of MTA configuration, only channel(s) marked with the `submit keyword` are capable of offering BURL support.

Technical Note
A client BURL command to an SMTP SUBMIT server for which BURL has not been enabled will be rejected with the error:

```
503 5.5.1 BURL has not been enabled.
```

BURL is only supported (may only be enabled) on SMTP SUBMIT channels; a client BURL command to an LMTP server will be rejected with the error:

```
503 5.5.0 BURL illegal on LMTP port.
```

while a client BURL command to an SMTP (non-SUBMIT) server will be rejected with the error:

```
503 5.5.1 BURL only allowed on submission port.
```

Through two different probe strings, the `BURL_ACCESS` mapping table controls whether the BURL extension is advertised, and then whether a client's BURL command is accepted. The first probe is performed before responding to an EHLO command. It has the form:

```
port_access-probe-info|channel|authentication-identity|
```

Here *`port_access-probe-info`* consists of all the information usually included in a `PORT_ACCESS` mapping table probe. It will be blank if BURL is being used in a "disconnected" context such as batch SMTP. *`channel`* is the current source channel. *`authentication-identity`* is the user's canonical authenticated login identity, normally *`uid@canonical-domain`* (that is, the user's LDAP *`uid`* attribute value, an at sign, and the canonical domain name in which the user's entry resides as found during domain lookup). *`authentication-identity`* will be blank if no authentication has been performed.

The "A" input flag will be set if SASL authentication has been performed, and the "T" input flag will be set if TLS is in use; thus the mapping templates may test using $:A and $:T, respectively. In order to offer BURL support, the mapping output for this first probe must set $Y; it may also optionally provide a space-separated list of supported URL types. imap is assumed if no explicit string is returned.

The second probe is performed when a BURL command is actually sent by the SUBMIT client and received by the SMTP SUBMIT server. In addition to the prior fields (described above), this second probe also includes as a final *client-url* field the URL specified in the client's BURL command:

```
port_access-probe-info|channel|authentication-identity|client-url
```

where *client-url* would normally be in URLAUTH syntax as defined in RFC 4467 (IMAP - URLAUTH Extension). See RFC 4467 for details on URLAUTH syntax, but for a rough idea to better understand the remainder of this discussion, consider that for BURL "submit user" access, the *client-url* will usually take a form along the lines of:

```
imap://enc-user@hostport/optional-expire-clause;URLAUTH=submit+enc-user:mec
```

(Other types of access in a URLAUTH and hence other forms of *client-url*, such as for "anonymous" access, are possible, but their use is more questionable--see the "Security Considerations" section of RFC 4468--and therefore the remainder of this discussion will focus on enabling only "submit user" access.)

In this second (actual BURL command) BURL_ACCESS probe, as in the prior (advertise BURL) probe, the "A" input flag will be set if SASL authentication has been performed, and the "T" input flag will be set if TLS is in use; thus the mapping templates may test using $:A and $:T, respectively. Additionally, for this second (actual received BURL command) probe, the vertical bar flag, |, will be set if the original URL sent by the client contained any vertical bars (which if present could possibly confuse some sorts of access checks), and thus the mapping entry template may test for vertical bars in the original client URL using the $:| test; note that the MAPPING_PARANOIA MTA option, if set, will cause any vertical bar within the client's original URL to be replaced in the probe by the specified alternate character (with the vertical bar input flag still being set). The mapping must set $Y for the URL to be accepted for processing. If $D is also set, then the string result of the mapping replaces the client's entire originally specified URL. Starting with Messaging Server 7 Update 4, if $M is set, then the IMAP host name in the client's original URL will be replaced by the mailHost of the currently authenticated user; this tends to provide both better security (by enforcing that BURL connections will only be made to a site's own mailHost host(s)), and better performance (by more likely connecting to the "correct" host), than relying on the IMAP host name supplied by the user's client. See BURL_ACCESS mapping flags table for a summary of available flags and tests.

BURL_ACCESS mapping flags

Flag	Description
$Y *url-types*	For the initial (EHLO) probe, enable advertising BURL support, optionally providing via the string argument `url-types` a space-separated list of the URL types to advertise.
$N *string*	® For the later (BURL command) probes, reject access. If the optional `string` is supplied, use it as the (entire) SMTP rejection, including SMTP error code and extended code as well as text. If no such `string` is supplied, then the default SMTP error used for such rejections is: 533 5.7.1 Access denied to specified URL.
$I	® For the later (BURL command) probes, if specified in an entry with `$N` rejecting that BURL command, the `$I` means further to forcibly disconnect the session with disconnect reason text "BURL_ACCESS forced disconnect". (Feature introduced in **Messaging Server 7 Update 4.**)
$Y	For the later (BURL command) probes, a plain `$Y` flag allows the BURL command.
$D *new-url*	® For the later (BURL command) probes, if `$Y` was also specified (allowing access), then use the specified `new-url` instead of the URL that the SMTP SUBMIT client provided.
$M	® For the later (BURL command) probes that succeed, in the actual BURL command override the IMAP host name in the client-provided URL and instead connect to the host of the `mailHost` attribute of the currently authenticated user. The BURL command probe itself will be considered to fail if the currently authenticated user has no `mailHost` attribute set.
$T	® For the later (BURL command) probes that succeed, force TLS use for opening the BURL URL (force TLS use in the connection to fetch the part specified in the client's BURL URL).
$X	® For the later (BURL command) probes that succeed, forcibly disable TLS use for opening the BURL URL (force non-use of TLS in the connection to fetch the part specified in the client's BURL URL).
$B	Disables certificate chain of trust validation for IMAPS: URLs and IMAP STARTTLS operations.

Flag comparisons	Description (These features introduced in Messaging Server 7 Update 4.)
$:|	® Match only if the original client URL included the vertical bar character, |.
$;|	® Match only if the original client URL included no vertical bar character, |.
$:A	Match only if SASL (SMTP AUTH) was used.
$;A	Match only if SASL (SMTP AUTH) was not used.
$:T	Match only if TLS was used.
$;T	Match only if TLS was not used.

®Available for the later (BURL command containing a URL) probes only; not available for the initial probe on whether or not to offer the BURL extension
---
At an absolute minimum, a site's `BURL_ACCESS` mapping table should be configured to verify that a proper type of URL has been specified: typically only `imap:` URLs should be allowed. Additionally, in the case of IMAP URLs used in SMTP SUBMIT message submission, a check ought to be made to insure that the URL "belongs" to the user: that is, that the access user in the URL matches the authenticated `uid` for the SUBMIT session. Indeed, typically sites will not even want to advertise BURL in the SMTP SUBMIT server response unless and until the client has authenticated. In terms of the `BURL_ACCESS` mapping table, this means to start with an entry that enables advertising BURL only if the *`authentication-identity`* field in the initial probe is non-empty. Additionally, it is almost always

essential to restrict message fetching access via a BURL command's `imap:` URL to an appropriate set of IMAP servers. Starting with Messaging Server 7 Update 4, use of the `$M` flag in the mapping template (right hand side) is a simple way to enforce that only users' own `mailHost` values are used as the IMAP server in the eventually executed BURL command. Prior to Messaging Server 7 Update 4, the `BURL_ACCESS` mapping table should typically be setup up to explicitly look for (match) a list of known, internal IMAP servers (users' valid `mailHost` values) and only allow BURL commands using those IMAP server host names.

Thus, starting with Messaging Server 7 Update 4, a minimal `BURL_ACCESS` mapping table might be something like:

```
BURL_ACCESS

! Initial entry to allow advertising BURL in EHLO response
*|tcp_*|%*| imap$Y
! Allow BURL commands, connecting to user's own mailHost
*|*@*|imap://*;URLAUTH=submit+$1*:* $:A$M$Y
```

Or if hosted domains are in use, then include an additional entry to match the hosted domain user use:

```
BURL_ACCESS

! Initial entry to allow advertising BURL in EHLO response
*|tcp_*|%*| imap$Y
! Allow BURL commands, connecting to (default domain) user's own
mailHost
*|*@*|imap://*;URLAUTH=submit+$1*:* $:A$M$Y
! Allow BURL commands, connecting to (hosted domain) user's own mailHost
*|*@*|imap://*;URLAUTH=submit+$1*%40$2*:* $:A$M$Y
```

A better minimal `BURL_ACCESS` mapping table, implementing the recommended security provisions of RFC 4468 (that is, also requiring TLS encryption as well as SMTP AUTH authentication in order to allow BURL), would be (starting Messaging Server 7 Update 4):

```
BURL_ACCESS

! Initial entry to allow advertising BURL in EHLO response, if TLS was
used
*|tcp_*|%*| $:T$Yimap
! Allow BURL commands, connecting to (default domain) user's own
mailHost
*|*@*|imap://*;URLAUTH=submit+$1*:* $:A$:T$M$Y
! Allow BURL commands, connecting to (hosted domain) user's own mailHost
*|*@*|imap://*;URLAUTH=submit+$1*%40$2*:* $:A$:T$M$Y
```

In versions prior to 7 Update 4, a minimal `BURL_ACCESS` mapping table would be more verbose. For instance, the following makes use of a subsidiary `X-IMAP-HOSTS` mapping table to list a site's valid IMAP server host names.

```
X-IMAP-HOSTS

mail1.example.com $Y
mail2.example.com $Y
mail3.example.com $Y

BURL_ACCESS

! Initial entry to allow advertising BURL in EHLO response
*|tcp_*|%*| imap$Y
! Allow BURL commands that request connecting to one of the "known" IMAP
hosts
! listed in the X-IMAP-HOSTS mapping table
*|*@*|imap://*@*/*;URLAUTH=submit+$1*:* $C$:A$|X-IMAP-HOSTS;$4|$Y$E
! Ditto for users in hosted domains
*|*@*|imap://*@*/*;URLAUTH=submit+$1*%40$2*:*
$C$:A$|X-IMAP-HOSTS;$4|$Y$E
```

Once the SMTP SUBMIT server has checked that BURL use should be allowed in a particular context, then in order to perform the `IMAP URLAUTH` operation specified in a BURL command, the SMTP SUBMIT server has to have the ability to log in to the IMAP server as the submit user. The `IMAP_USERNAME` and `IMAP_PASSWORD` MTA options are used to accomplish this. `IMAP_USERNAME` specifies the submit user and defaults to the setting of the `service.imap.submituser configutil` option if not specified. `IMAP_PASSWORD` specifies the password which of course much match the value set for the submit user account. The `IMAP_PASSWORD` option has no default value.

Technical Note
Once logged in as the submit user, then the BURL check of the hashed authentication token in the `URLAUTH` is performed: only messages accessible to the user issuing the BURL command will be fetched. That is, the submit user logs in, but it is not the submit user's message access that determines whether a message or message part can be fetched and incorporated but rather the access of the original user is validated from the `URLAUTH`.

When using BURL to fetch an entire, pre-composed message, a BURL command replaces the usual DATA command in an SMTP dialogue. Alternatively, when the SMTP extension CHUNKING is supported (see RFC 3030 and the chunking* channel keywords), then BURL and BDAT commands may be interlaced, meaning that a new message may be composed incorporating material (for instance, attachments) fetched via BURL commands. Unless explicitly disabled (see the `nochunkingserver` channel keyword), the MTA's TCP/IP channels (and in particular its SMTP SUBMIT servers) by default support CHUNKING.

Note that new in Messaging Server 7 Update 4, the MTA has a feature to force disconnection of the SMTP SUBMIT session if a user attempts "too many" bad (failed) BURL commands: see the `disconnectbadburllimit` channel keyword.

Note that as regards MTA message transaction logging, a message submitted using BURL will include a "U" modifier on its "E" enqueue record, or include "UC" if both BURL and CHUNKING were used. See MTA's transaction log entry format.

MTA Options for BURL

Option	Description
MAPPING_PARANOIA	(integer) Since address *_ACCESS mappings and the AUTH_REWRITE mapping use vertical bars as delimiters, issues can arise when externally provided material such as envelope From: or To: addresses contain vertical bars. This option is intended to provide various tools to handle such issues. Giving this option a non-negative value will cause any vertical bars in the externally supplied portions of various mapping input strings to be replaced with the character whose ASCII value is given by this option in the mapping probe. That is, the "regular," field-separating, vertical bars will still be present, but a vertical bar within an external field (such as within an address) will be replaced by the specified character. A negative value will cause the vertical bar to simply be dropped entirely from the probe. The default value for this option is 124, the ASCII value for vertical bar, which causes vertical bars to be left untouched. Note that many mapping tables where MAPPINGS_PARANOIA is relevant also have a feature for testing whether a vertical bar was originally present in externally supplied probe fields; use of MAPPING_PARANOIA to replace the original vertical bar characters does not affect such testing--the test flag is set based on the original presence of a vertical bar character, regardless of whether it is later replaced due to MAPPING_PARANOIA. This feature was introduced in **Message Server 7.**
IMAP_PASSWORD	(string) In order to perform an IMAP BURL operation, the SMTP SUBMIT server has to have the ability to log in to the IMAP server as the submit user. The IMAP_PASSWORD option specifies the password to use for such operations (and of course must match the value set for the submit user account). This option has no default. This feature was introduced in **Message Server 7.**
IMAP_USERNAME	(string) In order to perform an IMAP BURL operation, the SMTP SUBMIT server has to have the ability to log in to the IMAP server as the submit user. The IMAP_USERNAME option specifies the submit user; if not set, it defaults to the setting of the service.imap.submituser configutil parameter. This feature was introduced in **Message Server 7.**

Chapter 38. Messaging Server Lemonade Profile 1 Support

Lemonade Profile 1 Support

Support for most of Lemonade Profile 1 was introduced in **Messaging Server 7**. Absent from Messaging Server's support for Lemonade Profile 1 is SMTP BINARYMIME. In addition, Messaging Server's CONDSTORE and ANNOTATE implementations might cause performance issues, so use caution when working with these features. See the appropriate sections in the following information for details.

Introduction to Lemonade

Lemonade refers to an IETF working group formed to address the requirements of supporting standards-based email in a mobile or other resource-constrained environment. A "resource-constrained" environment is one where any or all of the following might be encountered:

- Low bandwidth, high latency networks
- Intermittent network connectivity
- Scarce power and compute cycles
- Minimizing data usage is a goal

The Lemonade Profile (RFC 4550, http://tools.ietf.org/html/rfc4550) defines a set of IMAP and SMTP extensions that address these constraints. Messaging Server implements most of the extensions defined in RFC 4550 (Lemonade Profile 1) and some of the extensions defined in RFC 5550 (Lemonade Profile 2). This page describes the configurable extensions.

Note
The Lemonade standard is mostly intended for mobile clients. Its goal is to reduce network traffic (both in volume and number of interactions) and to move the CPU load from the client to the server. Clients must have support for Lemonade built into them.

Lemonade Features

Some of the more interesting features of Lemonaded include the following:

- Forward a message without download (enabled by CATENATE, URLAUTH, and BURL)
- Quick resync (enabled by CONDSTORE and QRESYNC)

- Persistent sort and search (enabled by CONTEXT)
- Conversion (note that the only conversion supported in Messaging Server 7.0 is character set conversion)

The following sections describe these features in more detail.

Support for BURL

> **Note**
> Starting with Messaging Server 7 Update 4, refer to Enabling BURL Support for SMTP SUBMIT for details on configuring and using BURL.

Messaging Server 7.0 supports the BURL command, which extends the SMTP submission profile by adding a new command to fetch submission data from an IMAP server. This permits a mail client to inject content from an IMAP server into the SMTP infrastructure without downloading it to the client and uploading it back to the server. Thus, you could forward an email message without first downloading it to the client.

For more information, see http://www.ietf.org/rfc/rfc4468.txt.

Support is enabled in Messaging Server 7.0 by a new BURL_ACCESS mapping. The mapping receives two different probe strings:

```
port_access-probe-info|channel|uid|
port_access-probe-info|channel|uid|url
```

Here `port_access-probe-info` consists of all the information usually included in a PORT_ACCESS mapping table probe. It will be blank if BURL is being used in a "disconnected" context such as batch SMTP. The `channel` is the current source channel and `uid` is the user's authenticatd UID. The `uid` will be blank if no authentication has been performed. The `$:S` input flags will be set if SASL authentication has been performed and `$:T` will be set if TLS is in use.

The first probe is done when responding to EHLO. In order to offer BURL support the mapping must set `$Y` and optionally provide a space-separated list of supported URL types. The mapping assumes `imap` if no string is returned.

The second probe is performed when a BURL command is actually sent by the submit client. It includes the URL specified in the BURL command. Additionally, `$:|` will be set if the URL contains any vertical bars (which if present could possibly confuse some sorts of access checks). The mapping must set `$Y` for the URL to be accepted for processing. If `$D` is also set the string result of the mapping replaces the originally specified URL.

At an absolute minimum the mapping must verify that a proper type of URL has been specified. Typically only `imap:` URLs should be allowed. Additionally, in the case of "submit" IMAP URLs, a check needs to be made to insure that the URL belongs to the user, that is, the access user in the URL matches the authenticated UID for the submit session. Additionally, it is almost always essential to restrict access to an appropriate set of IMAP servers. So, assuming that all users connect to an IMAP server called `mail.example.com`, a minimal BURL_ACCESS would be something like:

```
BURL_ACCESS

  *|
imap$Y
  *|*@*|imap://$S*@mail.example.com/*;URLAUTH=submit+$1*:internal:$H*
$Y
```

The SMTP server has to have the ability to log in to the IMAP server as the submit user. The IMAP_USERNAME and IMAP_PASSWORD MTA options are used to accomplish this. IMAP_USERNAME specifies the submit user and defaults to the setting of the `service.imap.submituser` `configutil` option if not specified. IMAP_PASSWORD specifies the password which of course much match the value set for the submit user account. The IMAP_PASSWORD option has no default value.

IMAP URLAUTH Support

Messaging Server 7.0 supports the URLAUTH extension to IMAP and the IMAP URL Scheme (IMAPURL). This extension provides a means by which an IMAP client can use URLs carrying authorization to access limited message data on the IMAP server. An IMAP server that supports this extension indicates this with a capability name of "URLAUTH."

For more information, see http://www.ietf.org/rfc/rfc4467.txt.

IMAP CATENATE Support

Messaging Server 7.0 supports the CATENATE extension to IMAP, which extends the APPEND command to allow clients to create messages on the IMAP server that may contain a combination of new data along with parts of (or entire) messages already on the server. Using this extension, the client can catenate parts of an already existing message onto a new message without having to first download the data and then upload it back to the server.

For more information, see http://www.ietf.org/rfc/rfc4469.txt.

IMAP Conditional Store Operation Support

Messaging Server 7.0 supports IMAP Conditional Store Operations (CONDSTORE). IMAP CONDSTORE enables clients to coordinate changes to a common IMAP mailbox, for example, when multiple users are accessing shared mailboxes. The Conditional Store facility provides a protected update mechanism for message state information that can detect and resolve conflicts between multiple writing mail clients. The Conditional Store facility also allows a client to quickly resynchronize mailbox flag changes.

Warning
Use caution when enabling CONDSTORE with Convergence, as there might be degradation of IMAP performance. Our hope is to fix this problem in a future release. When this happens, we will remove this caution.

For more information, see http://www.ietf.org/rfc/rfc4551.txt.

IMAP ANNOTATE Support

Messaging Server 7.0 supports the ANNOTATE extension to IMAP, which permits clients and servers to maintain "meta data" for messages, or individual message parts, stored in a mailbox on the server. For example, you could use IMAP ANNOTATE to attach comments and other useful information to a message, or to attach annotations to specific parts of a message, marking them as seen or important, or a comment added.

> **Warning**
> Use caution when enabling ANNOTATE, as there might be degradation of IMAP performance. Our hope is to fix this problem in a future release. When this happens, we will remove this caution.

For more information, see http://www.ietf.org/rfc/rfc5257.txt. Of note in this document is Section 3.4, "Access Control," which summarizes access control restrictions, including the new ACL "n" right.

Controlling `IMAP CAPABILITIES` Vector

When migrating a multi-system deployment from Messaging Server 6.3 to 7.0, it's important that the systems advertise consistent IMAP extension sets, especially with respect to `CONDSTORE`. During migration you can configure 7.0 to display the same capability set as the older Messaging Server version, and you can turn on the new features on all back systems simultaneously. This feature is only of real significance with `CONDSTORE` and if you have lemonade-aware clients.

Note that you can also filter the initial capability vector advertised in the IMAP banner in the MMP.

Set the `configutil` options below to `TRUE` to control the `CAPABILITIES` vector:

IMAP Capability	configutil Option
IMAP4	service.imap.capability.imap4
IMAP4rev1	service.imap.capability.imap4rev1
ACL	service.imap.capability.acl
QUOTA	service.imap.capability.quota
LITERAL+	service.imap.capability.literal_plus
NAMESPACE	service.imap.capability.namespace
UIDPLUS	service.imap.capability.uidplus
CHILDREN	service.imap.capability.children
BINARY	service.imap.capability.binary
UNSELECT	service.imap.capability.unselect
SORT	service.imap.capability.sort
CATENATE	service.imap.capability.catenate
URLAUTH	service.imap.capability.urlauth
LANGUAGE	service.imap.capability.language
ESEARCH	service.imap.capability.esearch
THREAD=ORDEREDSUBJECT	service.imap.capability.thread_subject
THREAD=REFERENCES	service.imap.capability.thread_references
CONDSTORE	service.imap.capability.condstore
XSENDER	service.imap.capability.xsender
X-NETSCAPE	service.imap.capability.x_netscape
XSERVERINFO	service.imap.capability.xserverinfo
X-SUN-SORT	service.imap.capability.x_sun_sort
ANNOTATE-EXPERIMENT-1	service.imap.capability.annotate
X-UNAUTHENTICATE	service.imap.capability.x_unauthenticate
X-SUN-IMAP	service.imap.capability.x_sun_imap
X-ANNOTATEMORE	service.imap.capability.x_annotatemore
X-DRAFT-I01-EXPUNGED	service.imap.capability.x_expunged
XUM1	service.imap.capability.xum1
IDLE	service.imap.capability.idle
STARTTLS	service.imap.capability.starttls
XREFRESH	service.imap.capability.refresh

The default values for most of these options is TRUE. The exceptions are IMAP4 and CONDSTORE. The default for IMAP4 is FALSE unless local.obsoleteimap is set, in which case it is TRUE. These

options only affect whether a particular feature is advertised, except for `service.imap.capability.condstore` which also enables the feature.

Turning on (or not turning off) a capability does not necessarily mean that the feature will be advertised. `IDLE`, `STARTTLS`, and `XREFRESH` will only be advertised if enabled by these options and other condition exist that make it appropriate for them to be advertised.

Support for SMTP Submission Service Extension for Future Message Release

Messaging Server 7.0 supports Lemonade Profile 1, which is an extension to the SMTP submission protocol for a client to indicate a future time for the message to be released for delivery. This extension permits a client to use server-based storage for a message that should be held in queue until an appointed time in the future. This is useful for clients which do not have local storage or are otherwise unable to release a message for delivery at an appointed time. This functionality is useful for sending announcements to be read at the beginning of a work day, to send birthday greetings a day or so ahead, or to use as a lightweight facility to build a personal reminder service.

For more information, see http://tools.ietf.org/rfc/rfc4865.txt.

Support is enabled in Messaging Server by placing the `futurerelease`channel option on the source channel used for initial message submission. The keyword takes a single integer argument: the maximum number of seconds a message can be held.

Chapter 39. Messaging Server Monitoring Framework Support

Oracle Communications Messaging Server Monitoring Framework Support

> Support for Monitoring Framework is not present in **Messaging Server 7 Update 3**.

Messaging Server has supported the Sun Java System Monitoring Framework for a while (see the Sun Java Enterprise System 5 Monitoring Guide for details) to monitor various processes. Starting with Messaging Server 7.0, you can now specify the level of detail to monitor those processes. So, for example, in addition to monitoring how long each IMAP session lasts, you can also monitor individual IMAP transactions. This document how.

Topic:

- Setting Up and Using Monitoring Framework for Messaging Server

Setting Up and Using Monitoring Framework for Messaging Server

1) Install and configure the monitoring agent for Management Framework. See the Sun Java Enterprise System 5 Update 1 Monitoring Guide for details.

> **Note**
> This guide contains incorrect information. The procedure "To Enable Monitoring in Messaging Server" should specify the location of the `com.sun.cmm.ms.xml` file as: *msg-svr-base*`/install/configure_`*YYYYMMDDHHMMSS*`/com.sun.cmm.ms.xml`

2) Enable monitoring in Messaging Server. See Using the Monitoring Framework with Messaging Server.

3) Set the level of detail that that you want monitored.

This is done as follows:

```
configutil -o local.mfagent.monitor_level -v nnn
```

where *nnn* is an integer between 0 and 7. The larger the level, the more detail is monitored:

0 means no transaction monitoring
1 means process level monitoring
2 means session level monitoring
3 means message level monitoring
4 means fine level monitoring
5 means finest level monitoring

This parameter affect how much information the monitored processes provide to the `mfagent`.

The following other `configutil` options are read by the processes being monitored:

`Local.mfagent.port` (default 27433)
`Local.mfagent.secret` (no default)- note that this is mandatory.
`Local.mfagent.timeout` (default 30 seconds)
`Local.mfagent.listenaddr` (default INADDR_ANY)

Chapter 40. Triggering Effects From Transaction Logging. The LOG_ACTION Mapping Table

Triggering Effects From Transaction Logging: The LOG_ACTION Mapping Table

> **Note**
> The information on this page is generated automatically. Any changes made to this page will be overwritten the next time the information is updated. If you would like to request a change to this information, use the Comments feature. All comments will be considered.

(LOG_ACTION itself was added in MS 7.0 update 1. But use of LOG_ACTION with MeterMaid---other than simple "throttle" calls – typically requires MeterMaid features new in MS 7.0 update 2. In particular, MeterMaid "remove" and "test" routines are new in MS 7.0 update 2.) The LOG_ACTION mapping table provides a way for any transactions recorded by the MTA to also, as a side-effect, trigger other effects. A great deal of information, of different types, can be reported in the MTA's message transaction and connection transaction log files. Sites may be interested in noticing certain sorts of log entries as evidence of certain sorts of occurrences, or counting (or at least monitoring trends for) certain sorts of occurrences, or making access decisions based on certain sorts of occurrences. The LOG_ACTION mapping table provides a way to turn MTA message transaction and connection transaction log file entries into syslog notices, or into MeterMaid counter updates; or if a site wishes to provide their own routine for the mapping table to call, to take whatever, site-defined "action" the site chooses, based upon relevant transaction log entries. For instance, a site might want to notice (via a syslog notice) failed SMTP AUTH attempts as a warning of possible account break-in attempt; or a site might want to count (via MeterMaid) the number of failed (bad) recipients for users' outgoing messages, and react to "high" numbers as a possible sign of a user sending spam with a poor-quality recipient list.

LOG_ACTION operation

The LOG_ACTION mapping table is probed each time a message transaction or connection transaction log entry is written. The LOG_ACTION mapping table's only direct effect on the normal transaction log entries is its ability to disable output (recording) of specified entries. But its more interesting uses tend to be for its "side effects", which can be considered rather similar to the "side effects" available for the address based *_ACCESS mapping tables: In particular, it has the potential to generate syslog notices, or make call-outs such as to MeterMaid.

Probe format

The format of the probe for a message transaction log entry consists at a minimum of the following, plus additional, optional fields:

```
source-channel|destination-channel|action|size|envelope-from|orig-envelope-
```

Here *action* is the usually logged action code. Additional log entry fields may be included in the probe, depending upon the setting of the corresponding LOG_* MTA options; usually bit 1 (value 2) for a LOG_* MTA option controls whether the field controlled by that option is included in the LOG_ACTION probe. For LOG_CONNECTION, where bit 1 already has another meaning, bit 8 (value 256) enables inclusion of the claimed source system (as claimed in the client's the HELO/EHLO command for SMTP channels, or the enqueueing channel's official host name for other types of channels), and the bit that enables inclusion of *application-info* and *transport-info* in the LOG_ACTION probe is bit 9 (value 512); for LOG_INTERMEDIATE, bits 2 and 3 (values 4 and 8, respectively) control the inclusion of fields in the probe. These optional fields consist of:

```
|notary-bits|filename|envelope-id|message-id|username|source-system|sensiti
```

The *source-system*, and *application-info* and *transport-info* fields result from two bits of LOG_CONNECTION. The *intermediate-dest* and *initial-dest* fields result from two bits of LOG_INTERMEDIATE. The rest of the fields result from, respectively, LOG_NOTARY, LOG_FILENAME, LOG_ENVELOPE_ID, LOG_MESSAGE_ID, LOG_USERNAME, LOG_SENSITIVITY, LOG_PRIORITY, LOG_FILTER, LOG_REASON, LOG_DIAGNOSTICS, and LOG_QUEUE_TIME.

The format of the probe for a connection transaction log entry consists at a minimum of the following, plus additional, optional fields:

```
source-channel|direction|action
```

The *direction* is either + for inbound connections, or - for outbound connections. *action* is the usually logged connection action code, with the possible addition of an "F" suffix (on "C" or "X" action entries), added if the entry corresponds to a case where the MTA encountered an error with its attempt to create a *.data-failed file. The optional fields consist of any subset of:

```
|SASL-error-or-ETRN-host-name|username|diagnostics|transport-info|applicati
```

Optional fields are enabled by the relevant bit or bits of LOG_MESSAGE_ID (bit 1, value 2 enables logging of the SASL error in U SMTP AUTH records and the host name from client ETRN commands in I ETRN records), LOG_USERNAME (especially relevant for U SMTP AUTH records), LOG_DIAGNOSTICS, LOG_CONNECTION (bit 9, value 512 enables both *transport-info* and *application-info*), and LOG_QUEUE_TIME. Note that the same bit (or bits) enable probe inclusion both for message transaction probes and connection transactions probes.

If the probe string matches the pattern (*i.e.*, the left hand side of an entry in the mapping table), then the resulting output template (right hand side) is checked. The output templates in the LOG_ACTION mapping table can use the special flags defined in the table below, as well, of course, as any general mapping table substitutions or metacharacters such as calling out to a routine.

Table 25-3 LOG_ACTION mapping flags

Flag	Description
$F	Disable writing this entry to the transaction log file
$N	Disable writing this entry to the transaction log file
$< *string*	Send *string* as an OPCOM broadcast (OpenVMS) or to syslog (UNIX) or to the event log (NT) if probe matches; see also the SNDOPR_PRIORITY MTA option
$> *string*	Send *string* as an OPCOM broadcast (OpenVMS) or to syslog (UNIX) or to the event log (NT) if access is rejected; see also the SNDOPR_PRIORITY MTA option

Examples of LOG_ACTION use

This section shows examples of some possible uses of the LOG_ACTION mapping table. The syntax and general operation of the LOG_ACTION mapping table is discussed above. The first example below is a very simple use to disable recording of certain entries. The additional LOG_ACTION examples below are more sophisticated, making use of MeterMaid callouts.

Disabling logging of connections from a periodic monitoring source

One use of LOG_ACTION is to disable logging of some particular type of entry, while still retaining logging in general: for instance, connection attempts from some special source, when such connections are performed merely for "monitoring" or "heartbeat" reasons, may not deserve to be recorded.

For instance, if the monitor source IP is *monitor-ip*, then set bit 9 (value 512) of the LOG_CONNECTION MTA option and then use a LOG_ACTION mapping table along the lines of that shown below to disable the logging of the connection "O"pen and connection "C"lose records generated by the monitoring probe connections.

```
LOG_ACTION

  *|*|O|*|monitor-ip|*     $N
  *|*|C|*|monitor-ip|*     $N
```

Syslog notices after SMTP AUTH attempts with bad password

One use of LOG_ACTION might be to generate a syslog notice if more than some site-chosen number of bad password SMTP AUTH attempts are made against any user account. This can be achieved by calling out to MeterMaid from LOG_ACTION, and then generating the syslog notice when MeterMaid's threshold is reached.

First, in the MTA option file make sure that appropriate data will be included in the LOG_ACTION mapping table probes by setting LOG_* MTA options as below, and set SNDOPR_PRIORITY to values for syslog facility and severity that will coordinate with your `syslog.conf` configuration for syslog notice handling:

```
! Sites likely want additional bits of LOG_CONNECTION set; this example
! requires that bit 7/value 128 be set.
!
LOG_CONNECTION=128
LOG_MESSAGE_ID=3
LOG_USERNAME=3
LOG_DIAGNOSTICS=3
!
! Set SNDOPR_PRIORITY to a syslog facility+severity value that will
! coordinate with your syslog.conf configuration, e.g.
! SNDOPR_PRIORITY=20 to choose LOG_MAIL+LOG_WARNING syslog.conf
handling.
!
SNDOPR_PRIORITY=20
```

(The above settings represent likely site practice, rather than what is strictly required, as they show LOG_* option values set so that regular MTA connection transaction log entries will be generated as well as fields in LOG_ACTION probes; but in principle, the LOG_ACTION probes could be set without having to enable the connection transaction logging.)

To have a MeterMaid table named bad_password_attempts, configure MeterMaid via `configutil` values, including these (though note that many of the values shown being set are actually defaults):

```
metermaid.table.bad_password_attempts.data_type: string
metermaid.table.bad_password_attempts.max_entries: 1000
metermaid.table.bad_password_attempts.options: nocase,penalize
metermaid.table.bad_password_attempts.type: throttle
metermaid.table.bad_password_attempts.quota: 5
metermaid.table.bad_password_attempts.quota_time: 3600
```

(Additional, basic MeterMaid configuration will be needed via additional `configutil` parameters not shown above.)

Then a LOG_ACTION mapping table to make use of that MeterMaid table could be as follows:

```
LOG_ACTION

! With LOG_CONNECTION=128 set, we get "U" action records.
! With LOG_MESSAGE_ID=3 set, the SASL-error is recorded in the
message-id field
! With LOG_USERNAME=3, get a username field
! With LOG_DIAGNOSTICS=3, get a diagnostics field
!
! So probe format is:
!
! source-chan|direction|action|SASL-error|username|diagnostics
!
  tcp_*|+|U|Bad$ password|$_*|*   \
$[IMTA_LIB:check_metermaid.so,throttle,bad_password_attempts,$1]$<LOGACTION
\
Too$ many$ bad$ password$ attempts$ for$ user$ $1
```

There is a subtlety in the above, which is that LOG_DIAGNOSTICS is shown being set *purely* to ensure that there is at least one more vertical bar and (possibly empty) field appearing after the *username* field. Furthermore, asterisk wildcard for matching the *username* field has the "minimal matching" $_ modifier set on it.) This is not strictly necessary for this *exact* example, but makes this example easier to extend with additional fields should sites wish to do so.

Syslog notices after SMTP AUTH attempts with bad username

Another example would be to generate a syslog notice if the same source IP makes multiple attempts to authenticate with a bad username (hence suggestive of a possible "dictionary attack" against your user name space). With MTA options settings of:

```
! Sites likely want additional bits of LOG_CONNECTION set; this example
! requires that bit 1/value 2 plus bit 7/value 128 plus bit 9/value 512
be set.
LOG_CONNECTION=642
LOG_MESSAGE_ID=3
LOG_USERNAME=3
LOG_DIAGNOSTICS=3
!
! Set SNDOPR_PRIORITY to a syslog facility+severity value that will
! coordinate with your syslog.conf configuration, e.g.
! SNDOPR_PRIORITY=20 to choose LOG_MAIL+LOG_WARNING syslog.conf
handling.
!
SNDOPR_PRIORITY=20
```

and `configutil` MeterMaid table settings that, beyond basic settings not shown, include:

```
metermaid.table.bad_password_attempts.data_type: ipv4
metermaid.table.bad_password_attempts.max_entries: 1000
metermaid.table.bad_password_attempts.options: penalize
metermaid.table.bad_password_attempts.type: throttle
metermaid.table.bad_password_attempts.quota: 5
metermaid.table.bad_password_attempts.quota_time: 3600
```

then a LOG_ACTION mapping table could be:

```
LOG_ACTION

! With LOG_CONNECTION=642 set, we get "U" action records and
transport-info
! fields.
! With LOG_MESSAGE_ID=3 set, the SASL-error is recorded in the
message-id field
! With LOG_USERNAME=3, get a username field
! With LOG_DIAGNOSTICS=3, get a diagnostics field
!
! So probe format is:
!
!
source-chan|direction|action|SASL-error|username|diagnostics|transport-info

! where transport-info is:  TCP|host-IP|host-port|source-IP|source-port
!
  tcp_*|+|U|No$ such$ user|*|*|TCP|$_*|$_*|$_*|*   \
$[IMTA_LIB:check_metermaid.so,throttle,bad_user_attempts,$5]$<LOGACTION,$
\
Too$ many$ wrong$ username$ attempts$ from$ $5$ (username$ attempted:$
$1)
```

Syslog notices after failing SMTP AUTH attempts, resetting after success

Both of the above examples could be improved by using the (new in 7.0 update 2) `remove` routine of MeterMaid to achieve a "reset" effect after desired "good" occurrences. For instance, with settings as above (including LOG_DIAGNOSTICS=3 and LOG_CONNECTION=642):

```
LOG_ACTION

! With LOG_CONNECTION=642 set, we get "U" action records and
transport-info
! fields.
! With LOG_MESSAGE_ID=3 set, the SASL-error is recorded in the
message-id field
! With LOG_USERNAME=3, get a username field
! With LOG_DIAGNOSTICS=3, get a diagnostics field
!
! So probe format is:
!
!
source-chan|direction|action|SASL-error|username|diagnostics|transport-info

! where transport-info is:  TCP|host-IP|host-port|source-IP|source-port
!
  tcp_*|+|U|Bad$ password|$_*|*    \
$[IMTA_LIB:check_metermaid.so,throttle,bad_password_attempts,$1]$<LOGACTION
\
Too$ many$ bad$ password$ attempts$ for$ user$ $1
  tcp_*|+|U|No$ such$ user|*|*|TCP|$_*|$_*|$_*|*    \
$[IMTA_LIB:check_metermaid.so,throttle,bad_user_attempts,$5]$<LOGACTION,$
\
Too$ many$ wrong$ username$ attempts$ from$ $5$ (username$ attempted:$
$1)
!
! Once a successful AUTH occurs, remove the entry for that username in
the
! bad_password_attempts table, and remove the entry for that source IP
in
! the bad_user_attempts table.  We want to try to remove the source IP
entry in
! the second table, even if (for some reason) the MeterMaid callout on
the
! first fails, so we split the store calls into two separate entries,
rather
! than attempting two routine calls from one right hand side.
!
  tcp_*|+|U|*|*|*authentication$ successful*|TCP|$_*|$_*|$_*|*
$CCLEAR|$2|$7
!
! If the authentication was successful, then the probe is now:
! CLEAR|username|source-ip
!
  CLEAR|*|*  \
$CCLEAR|$0|$1$[IMTA_LIB:check_metermaid.so,remove,bad_password_attempts,$0]

  CLEAR|*|*
$[IMTA_LIB:check_metermaid.so,remove,bad_user_attempts,$1]
```

In the above example, it is convenient to detect successful authentication via the diagnostics field (the SMTP response). Attempting to detect successful authentication via the reason field would be more

awkward, as the reason field may be either "`Switched to channel` *`channel-name`*" when authentication succeeded and resulted also in a channel switch, or the reason field will be empty when authentication succeeded but no channel switch was performed, but this can be confounded with certain failure cases that also result in an empty reason field.

Syslog notices when time-in-queue becomes "high", ceasing after any quick delivery

(Note that this example uses the MeterMaid "remove" routine, new in MS 7.0 update 2.) Another example of generating syslog notices when some condition occurs, and then resetting the MeterMaid table entry (and hence stopping the syslog notices) when the condition ceases, would be for cases where messages, while getting delivered upon first delivery attempt, are not getting delivered sufficiently promptly for the site's taste. That is, considering only messages that actually do manage to get delivered upon first attempt rather than failing a delivery attempt and having to be retried later (hence inherently taking a relatively "long" time to be delivered), the site wishes to watch for cases where that initial, successful, delivery attempt nevertheless was rather "slow". Here we will record in MeterMaid the destination domain for each new ZZ* message whose delivery is "slow" (taking 300 seconds or more), and generate a syslog notice for such domains once 5 messages have been slow within an hour (3600 seconds), but reset (to 0) the MeterMaid table entry for that domain once a "quick delivery" (within 60 seconds) has occurred.

```
LOG_FILENAME=3
LOG_QUEUE_TIME=3
!
! Set SNDOPR_PRIORITY to a syslog facility+severity value that will
! coordinate with your syslog.conf configuration, e.g.
! SNDOPR_PRIORITY=20 to choose LOG_MAIL+LOG_WARNING syslog.conf
handling.
!
SNDOPR_PRIORITY=20
```

```
metermaid.table.slow_delivery.data_type: string
metermaid.table.slow_delivery.max_entries: 2000
metermaid.table.slow_delivery.options: nocase
metermaid.table.slow_delivery.type: throttle
metermaid.table.slow_delivery.quota: 5
metermaid.table.slow_delivery.quota_time: 3600
```

```
LOG_ACTION

!
source-chan|dest-chan|D|size|env-from|orig-env-to|env-to|filename|queue-tim

!
  *|*|D*|*|*|*|*@*|ZZ*|*   $CDEQUEUEDOMAINTIME|$7|$9
!
! Now probing with just DEQUEUEDOMAINTIME|domain|queue-time
!
  DEQUEUEDOMAINTIME|*|*    $CFASTDOMAIN|$`integer($1)<=60'|$0|$1
!
! Now the probe will be FASTDOMAIN|1|domain|queue-time for
queue-time<=60s
! or FASTDOMAIN|0|domain|queue-time for queue-time>60
!
  FASTDOMAIN|1|*|*         \
$[IMTA_LIB:check_metermaid.so,remove,slow_delivery,$0]
  FASTDOMAIN|0|*|*         $CSLOWDOMAIN|$`integer($1)>=300'|$0|$1
!
! Now the probe will be SLOWDOMAIN|1|domain|queue-time for
queue-time>=300s
! or SLOWDOMAIN|0|domain|queue-time for 60s<queue-time<300s
!
  SLOWDOMAIN|1|*|*         \
$[IMTA_LIB:check_metermaid.so,throttle,slow_delivery,$0]$<LOGACTION,$ \
Domain$ $0$ deliveries$ slow
```

The above example uses the mapping table expression substitution syntax, $`*expression*', to perform a test on the value of the *queue-time* (time in queue) field: such an expression will evaluate to 1 if true, 0 if false, so such an expression substitution inserts the string 1 if the numeric test was true or inserts the string 0 if the numeric test was false.

Blocking submissions of local senders who may be spammers

(Note that this example uses the MeterMaid "test" routine, new in MS 7.0 update 2.) A different type of use of LOG_ACTION is to update a MeterMaid table with information relating to message dequeue, which a message enqueue access mapping table can then consult. That is, LOG_ACTION with MeterMaid can provide a way for decisions on allowing message enqueue to reference historical data regarding message dequeue. For instance, a local user whose outbound to the Internet messages suffer a high number of rejections of bad addresses by remote destinations may raise the suspicion that that user is attempting to send spam using a poor quality recipient list; a site may wish to deny that account further submissions that day.

For any such access restrictions on senders, it is always strongly advisable to *require* local users to authenticate (use SMTP AUTH) in order to submit messages, and then to perform the access checks using the *authenticated* submission address. Because the SMTP envelope From is easy to forge in the base SMTP protocol, attempts to enforce access restrictions based solely upon envelope From are regrettably likely to unintentionally *encourage* (or at least motivate) users to forge their envelope From as a way to bypass the access restrictions. Thus poorly considered access restrictions on senders can cause more harm than good, by motivating users to obscure their identity to evade the restrictions. So this example will assume that a security-conscious configuration, where users are required to use SMTP

AUTH to submit, is already in place, so that the authenticated username is available both for logging via the LOG_USERNAME MTA option, and for performing an access check from the FROM_ACCESS mapping table.

So with MTA options settings that include:

```
! Bit 1/value 2 is not set in any of:
!  LOG_NOTARY, LOG_FILENAME, LOG_ENVELOPE_ID, or LOG_MESSAGE_ID
LOG_REASON=3
LOG_USERNAME=3
LOG_DIAGNOSTICS=3
```

and `configutil` MeterMaid table settings that, in addition to basic configuration not shown here, include:

```
metermaid.table.sends_to_bogus_recipients.data_type: string
metermaid.table.sends_to_bogus_recipients.max_entries: 5000
metermaid.table.sends_to_bogus_recipients.options: nocase,penalize
metermaid.table.sends_to_bogus_recipients.type: throttle
metermaid.table.sends_to_bogus_recipients.quota: 15
metermaid.table.sends_to_bogus_recipients.quota_time: 86400
```

then a LOG_ACTION mapping table to track in MeterMaid bad recipient addresses discovered at dequeue time, and a corresponding FROM_ACCESS mapping table that checks that MeterMaid data to decide whether to allow new message submissions, could be:

```
LOG_ACTION

! source-chan|dest-chan|action|size|env-from|orig-env-to|env-to|
!  username|reason|diagnostics
!
  tcp_local||R*|*|*|*|*|$**|Remote$ SMTP$ server$ has$ rejected$
address|*  \
$[IMTA_LIB:check_metermaid.so,throttle,sends_to_bogus_recipients,$5]
  tcp_local||K*|*|*|*|*|$**|Remote$ SMTP$ server$ has$ rejected$
address|*  \
$[IMTA_LIB:check_metermaid.so,throttle,sends_to_bogus_recipients,$5]

FROM_ACCESS

  TCP|*|*|*|*|SMTP*|MAIL|tcp_auth|*|*   \
$[IMTA_LIB:check_metermaid.so,test,sends_to_bogus_recipients,$6,>=15]$NYou$
\
have$ sent$ to$ too$ many$ bad$ addresses$ today
```

Note that since in the logging (MTA transaction log entries and hence the LOG_ACTION field) the username field resulting from SMTP AUTH authentication is actually the `mail` attribute value with the asterisk character prefixed, whereas in FROM_ACCESS the "username" field is simply the pure `mail` attribute value, above in LOG_ACTION the entries make sure to explicitly match the asterisk character

and then *not* include it in the sends_to_bogus_recipients table updates, so that the FROM_ACCESS probes can match on just the username.

Chapter 41. Using Role-Based Access Control in Messaging Server

Using Role-Based Access Control in Oracle Communications Messaging Server

This information describes role-based access control and the required setup for Oracle Solaris 10 where privileges are available.

Topics:

Overview of Role-Based Access Control

Role-based access control (RBAC), a feature in Oracle Solaris, permits non-privileged users to have access to certain privileged functionality, under certain specified circumstances. At a minimum, you can grant the equivalent of `setuid root` to a particular program, but only when run by a certain user. RBAC enables you to fine-tune access to privileges so that they are available in a restricted environment and only when needed.

In addition, Oracle Solaris 10 includes privileges that give finer-grained access so that a process that requires elevated access can be granted just the minimum access necessary to satisfy its needs without having use the traditional `UID 0` full-access. For example, a program that needs to bind to a privileged port (typically one with a port number that is less than 1024, such as port 25 for SMTP) would have needed `root` access just for that one activity. With privileges, the program can use the `net_privaddr` privilege to grant it the access needed to bind to the port without having full root access. By compartmentalizing privileged functions, security is greatly enhanced.

You can use RBAC for both methods, and each improves Messaging Server security.

Theory of Operations

Role-based access control is managed through several files that are located in the `/etc` and `/etc/security` directories. You first create a profile that defines the new access that can be granted to the Messaging Server user account. Then you list all the special access that is granted to that profile. Finally, the Messaging Server user account is given access to the new profile.

The special access permitted by the profile is managed through intermediate commands that run the programs with the defined access. The `pfexec(1)` command is generally responsible for running a program that can then be given elevated access. Starting with Messaging Server 7 Update 2, `pfexec` is used by `start-msg`, `stop-msg`, `imsimta` (through the `imtacli` program), and the `job_controller` to take advantage of role-based access controls.

For more information about role-based access controls, see `rbac(5)`.

Pre-Deployment Preparations

Although RBAC is available on Oracle Solaris 8 and 9, the example files included with Messaging Server 7 Update 2 are designed for Oracle Solaris 10 and take advantage of the availability of privileges, which are new in Oracle Solaris 10.

Setting Up and Using RBAC

Caution
Use caution when implementing role-based access controls as it involves modifying system files that provide security definitions for the operating system, and incorrect modifications may result in potential problems.

The following steps make direct modifications to files in the `/etc/security` directly, which can also be made by using the Oracle Solaris Management Console (`smc(1m)`).

Assumptions in the Examples
The following example commands assume that the Messaging Server was installed in the `/opt/sun/comms/messaging64` directory and that you have chosen `mailsrv` as the Unix user used by the Messaging Server processes.

1. Copy *msg-svr-root*`/examples/rbac/RtSJSMessagingServer.html` to the `/usr/lib/help/profiles/locale/C` directory.
   This file is referenced by the Messaging Server profile definition. For Messaging Server releases prior to Release 7 Update 4, perform the following:

   ```
   # cp /opt/sun/comms/messaging64/examples/rbac/RtSJSMessagingServer.html
   /usr/lib/help/profiles/locale/C
   ```

   Starting with Messaging Server Release 7 Update 4, perform the following:

   ```
   # cp /opt/sun/comms/messaging64/examples/rbac/MessagingServer.html
   /usr/lib/help/profiles/locale/C
   ```

2. Append the contents of *msg-svr-root*`/examples/rbac/prof_attr.example` to `/etc/security/prof_attr`.
   This is the Messaging Server profile definition.

   ```
   # cat /opt/sun/comms/messaging64/examples/rbac/prof_attr.example >>
   /etc/security/prof_attr
   ```

3. Edit *msg-svr-root*`/examples/rbac/exec_attr.example` to replace `<msg.RootPath>` with the actual path for your Messaging Server installation.
   For this example, instances of `<msg.RootPath>` are replaced with `/opt/sun/comms/messaging64`.
4. Append the contents of the edited *msg-svr-root*`/examples/rbac/exec_attr.example` to `/etc/security/exec_attr`.

This defines the special permissions granted to the Messaging Server profile.

```
# cat /opt/sun/comms/messaging64/examples/rbac/exec_attr.example >>
/etc/security/exec_attr
```

5. Modify the user account used by the Messaging Server to have access to this new profile.
   For Messaging Server releases prior to Release 7 Update 4, perform the following:

   ```
   # usermod -P 'Sun Java System Messaging Server' mailsrv
   ```

   Starting with Messaging Server Release 7 Update 4, perform the following:

   ```
   # usermod -P 'Oracle Communications Messaging Server' mailsrv
   ```

6. For Messaging Server Release 7 Update 4, modify the dispatcher process privilege, so that the dispatcher is able to successfully start.
   Edit the `/etc/security/exec_attr` file and add `proc_taskid`, for example:

   ```
   Oracle Communications Messaging
   Server:solaris:cmd:::/opt/sun/comms/messaging64/lib/dispatcher:privs=net_
   ```

7. Beginning with Messaging Server 7 Update 3, set the `local.rbac configutil` option to 1 to fully enable RBAC usage.

New Behavior

Once the RBAC has been set up, the Messaging Server user has sufficient access so as not to require being run as `root`, to use the following commands:

- `start-msg`
- `stop-msg`
- `imsimta restart | shutdown | startup | stop`

Reference Information

For more information about role-based access controls, see the following sources:

- Oracle Solaris 10 documentation: System Administration Guide: Security Services (Roles, Rights Profiles, and Privileges)
- man pages: `smc(1M)`, `usermod(1M)`, `prof_attr(4)`, `exec_attr(4)`, `privileges(5)`, `rbac(5)`

Chapter 42. Using Service Management Framework with Messaging Server

Using Service Management Framework (SMF) with Messaging Server

> Beginning with the **Messaging Server 7**, SMF support has been integrated into the product. Messaging Server provides a single SMF service definition file.

SMF was added in Solaris OS 10 as a replacement to the `/etc/init.d` scripts for starting, stopping, and restarting services. SMF dramatically decreases boot time as it is aware of dependencies between services, and starts services in parallel where possible.

```
<msg_base>/data/install/messaging.xml
```

The SMF service definitions can be imported using the `svccfg` command.

```
svccfg import <msg_base>/data/install/messaging.xml
```

The following example shows how to check initial Messaging Server status, enable SMF, then verify status. Please note that Messaging Server must be stopped prior to using the `svcadm enable` command.

```
# svcs messaging_server

STATE          STIME    FMRI
disabled        8:58:29 svc:/network/messaging_server:default

# svcadm enable messaging_server

# svcs messaging_server
STATE          STIME    FMRI
online          9:08:54 svc:/network/messaging_server:default
```

For more information on SMF, see Managing Services (Overview), in the *Solaris System Administration Guide*. This chapter provides an overview SMF, including SMF concepts, administrative and programming interfaces, components, and run levels provided.

Chapter 43. Third-Party Authentication Server Support

Third-Party Authentication Server Support

> **Note**
> Messaging Server support for third-party authentication servers that validate and modify Messaging Server authentication information is available beginning with Messaging Server 7 Update 4 (**Patch 22**).

Topics:

Messaging Server provides support for third-party authentication servers by supporting a protocol designed to integrate third-party authentication services. This protocol is documented in the file `authserver.txt`, which is installed as part of the Messaging Server in the `examples/tpauthsdk` directory.

Support for third-party authentication servers addresses two primary problems:

- If your computing infrastructure does not store passwords in LDAP, Messaging Server can be configured to query the authentication server you provide to verify passwords. This circumvents the need to replicate passwords from the third-party authentication service to the LDAP server used by Messaging Server for user information.
- If you wish to use an authentication system that provides security or management capabilities not possible with traditional password authentication (such as Kerberos), Messaging Server can be configured to pass SASL mechanisms to the authentication server you provide for processing. While Messaging Server does not directly support Kerberos, it is now possible to write code that adds Kerberos capability to Messaging Server.

The `examples/tpauthsdk` directory also contains sample code for a third-party authentication server that can validate and modify authentication information provided by the Messaging Server.

Messaging Mutiplexor (MMP) Support

To enable third-party authentication in the MMP, the `PreAuth` configuration option must be enabled. For authentication methods other than PLAIN, you must specify the proxy authentication credentials (`StoreAdmin` and `StoreAdminPass`) and add the following configuration option to the MMP's `ImapProxyAService.cfg` and/or `PopProxyAService.cfg` file:

```
default:AuthenticationServer :56
```

For plain text logins, the MMP will first perform a normal user lookup in LDAP. Once the user is located, the MMP will connect to the host (localhost loopback) and port (56) specified in the `AuthenticationServer` option to authenticate the user. The MMP will pass the LDAP attributes

inetUserStatus, mailUserStatus, uid and mailHost to the authentication server. You may also configure the MMP to look up additional LDAP attributes and pass them to the authentication server with the option:

```
default:AuthenticationLdapAttributes "attr1" "attr2" ...
```

Note that the authentication server should be running on the same server as the MMP and on a restricted port. The protocol used to communicate between the MMP and the third-party authentication server is not presently secured.

The MMP will advertise the additional SASL mechanisms provided by the authentication server via the standard server protocols (for example, through the IMAP capabilities). The authentication server can provide a SASL mechanism that the MMP implements natively. In this case, the authentication server mechanism will take precedence. To see a transcript of the communication between the MMP and authentication server process, add authserv to the default:debugkeys configuration setting for the MMP and set default:LogLevel to Debug. State transitions in the HULA authentication subsystem can also be logged via the hula debug key.

See Configuring and Administering Multiplexor Services for details on configuring and administering MMP.

IMAP/POP/SMTP Support

To enable third-party authentication with IMAP/POP/SMTP, use the configutil options sasl.default.authenticationserver and sasl.default.authenticationldapattributes. These work as they do for the MMP except the authenticationattributes is a space-delimited list (quotes are not used). The local.debugkeys option provides functionality equivalent to the default:debugkeys option, but again as a space separated list.

To debug third-party authentication with store, set local.debugkeys to include authserv (or authserv hula) and set logfile.{imap/pop}.loglevel to debug.

Sample Code

The third-party authentication sample code in the tpauth directory is largely suitable for a production environment (it uses a thread-pool model to handle a high volume of connections). However, the Messaging Server product team will not provide support for this sample code. The sample code is designed for use on a system which provides the standard Posix Threads API (for example, Oracle Solaris or Linux). The Makefile.sample is for use on Oracle Solaris.

The table that follows lists the contents of the tpauth directory.

File	Contents
README.txt	A link to this wiki page.
authserver.txt	Third-Party Authentication Protocol Specification.
Makefile.sample	Use `make -f Makefile.sample` to build the sample code.
authserv.c	The core thread-pool protocol server implementation.
authserv.h	The API called by `authserv.c` to authenticate users.
sample.c	A very simple sample third-party authentication module using plain-text passwords. Third-parties may edit or replace this module to provide authentication services.
sample2.c	A simple CRAM-MD5 example demonstrating use of this interface for non-plain-text mechanisms.

Chapter 44. Messaging Server 8.0 32-bit Anti-Spam and Virus Plug-ins

Messaging Server 8.0 32-bit Anti-Spam and Virus Plug-ins

Messaging Server is now a native 64-bit application that takes full advantage of the power of Oracle Solaris 10 and 11.

The optimal way for third-party spam and virus filter vendors, such as Symantec or Cloudmark, to integrate with Messaging Server 8.0 is through a native 64-bit plug-in. Not all vendors, however, have completed their upgrade from 32-bit to 64-bit.

If your third-party spam or anti-virus vendor is not currently 64-bit ready, you have the following option:

Use the Messaging Server's Milter Interface
Many mail filtering products support connections from mail servers using the "milter" protocol. Milter is commonly used by Sendmail mail servers. If your filtering product supports milter, you can use the Messaging Server's milter interface. Milter does not provide all the functionality of a native 64-bit spam filter plug-in. For example, Milter does not allow users to be grouped into different classes of service for filtering or viruses to be removed from messages in flight.

Chapter 45. Configuring check_metermaid.so Clients to Access Multiple MeterMaid Servers

Configuring check_metermaid.so Clients to Access Multiple MeterMaid Servers

Topics:

Considerations for Distributing Load Across Multiple MeterMaid Servers

If you have multiple MeterMaid tables in your deployment, you may be able to improve overall performance by distributing them across multiple MeterMaid servers. The `check_metermaid.so` client supports associations between MeterMaid tables and the servers that are responsible for their respective tables. The client also supports per server options for concurrency and use of SSL.

Configuring check_metermaid.so to Access Multiple MeterMaid Servers

To configure `check_metermaid.so` to access multiple MeterMaid Servers:

1. Define the list of all tables and associate them with nicknames for each server using `msconfig` (Unified Configuration):

   ```
   set metermaid_client.remote_table:table1.server_nickname "alpha"
   set metermaid_client.remote_table:table2.server_nickname "alpha"
   set metermaid_client.remote_table:table3.server_nickname "beta"
   ```

   or using `configutil` (legacy configuration):

   ```
   configutil -o
   metermaid.mtaclient.remote_table.table1.server_nickname -v "alpha"
   configutil -o
   metermaid.mtaclient.remote_table.table2.server_nickname -v "alpha"
   configutil -o
   metermaid.mtaclient.remote_table.table3.server_nickname -v "beta"
   ```

   where *table1*, *table2*, and *table3* are tables defined in your deployment's MeterMaid configuration. The servers' nicknames allow the `check_metermaid.so` client to look for remote servers associated with those nicknames. Since nicknames become part of the configuration option names for the remote server definitions, they must include only letters, numbers, and underscores.

2. Create configuration entries for each server nickname for the host name and port for the server, the maximum number of connections, and whether it uses SSL. Run the following commands if you are using `msconfig` (Unified Configuration):

```
set metermaid_client.remote_server:alpha.max_conns 3
set metermaid_client.remote_server:alpha.server_host
"alpha.example.com"
set metermaid_client.remote_server:alpha.server_port 63837
set metermaid_client.remote_server:alpha.sslusessl 0
set metermaid_client.remote_server:beta.max_conns 3
set metermaid_client.remote_server:beta.server_host
"beta.example.com"
set metermaid_client.remote_server:beta.server_port 63837
set metermaid_client.remote_server:beta.sslusessl 0
```

Or run the following commands if you are using `configutil` (legacy configuration):

```
configutil -o metermaid.mtaclient.remote_server.alpha.max_conns -v
3
configutil -o metermaid.mtaclient.remote_server.alpha.server_host
-v "alpha.example.com"
configutil -o metermaid.mtaclient.remote_server.alpha.server_port
-v 63837
configutil -o metermaid.mtaclient.remote_server.alpha.sslusessl -v
0
configutil -o metermaid.mtaclient.remote_server.beta.max_conns -v 3
configutil -o metermaid.mtaclient.remote_server.beta.server_host -v
"beta.example.com"
configutil -o metermaid.mtaclient.remote_server.beta.server_port -v
63837
configutil -o metermaid.mtaclient.remote_server.beta.sslusessl -v 0
```

For descriptions of these options, please see the MeterMaid Reference.

3. If you are using Unified Configuration and your configuration is compiled, recompile your configuration. This step is not necessary if you are using legacy configuration:
   `imsimta cnbuild`
4. Restart components that are using `check_metermaid.so`. You would typically do this by restarting the dispatcher:

```
stop-msg dispatcher
start-msg dispatcher
```

Chapter 46. Managing Sieve Scripts

Managing Sieve Scripts

This document provides information about Oracle Communications Messaging Server's implementation and configuration of ManageSieve, a protocol for user management of Sieve mail filters (RFC 5228) specified in RFC 5804.

Location of Managed Sieves

The ManageSieve server manages a collection of Sieve scripts belonging to each user. You may select one of these scripts as the active script to filter the user's mail. You can store the scripts either in LDAP or in disk files. In either case, the active script is stored separately from the Sieve collection in either a different attribute or a separate file. You can store the script collection and the active script in different places. For example, you can store the collection on disk and store the active script in the user's LDAP entry.

Active scripts stored in LDAP can consist of multiple values of the same attribute. When the ManageSieve server reads multiple values, it appends them to form a single script. (Note, however, that you lose this storage pattern if the ManageSieve server writes the active script. The server always writes scripts to LDAP as a single attribute value.) The server stores the script collection differently. It uses a single attribute and takes each value to be a separate script.

The server stores active scripts on disk in a single file with a specific name. The server stores each script in the script collection in a separate file. Each file has the same name prefix, but the server appends a unique identifier to create a different file name for each script.

Script Name Storage

Each script in the collection has a name. The server stores this name as a comment at the beginning of the script. The ManageSieve server maintains this comment which does not appear as part of the Sieve data provided to the client.

The server does not store the active script's name in the script itself and therefore the script does not contain an initial comment.

Stored Script Semantics

The ManageSieve server is designed to accommodate the behavior of other agents that both access and modify stored Sieve scripts. It does not assume that stored scripts will remain unaltered and does not cache scripts internally.

Specifically, each ManageSieve operation causes the following initial steps to be performed:

1. The server reads the active script and script collections from their respective locations. It silently ignores scripts in the collection that do not begin with a valid name comment.
2. The server computes a hash for the content of each script that is found. Note that this hash does not cover the initial comment containing the script name.
3. The server performs a sanity check on all scripts in the collection. If two scripts have the same hash value, the server changes one of them by adding sufficient trailing white space to obtain a unique hash. Similarly, the server checks the names of every script in the collection. If two scripts have the same name, the server adds `-`*n*, where *n* is an integer, to make the name unique.
4. The server checks the hash of each script in the collection against the hash of the active script. The server considers the script with the matching hash to be the active script. If no script in the collection has a matching hash, the server creates a new script with a name of the form `ACTIVE-`*datetime*, where *datetime* is the time the server performs the operation.
5. The server writes immediately to LDAP, disk, or both any changes the preceding two steps made to the collection.

The server performs the requested operation only after it performs these steps. This operation may cause additional changes that the server writes when the operation finishes.

This rather convoluted approach ensures that the ManageSieve server's view of the user's Sieve state remains consistent even if other agents manipulate the underlying data in unexpected ways.

ManageSieve Service Configuration

ManageSieve runs as a service under the dispatcher. To configure the ManageSieve service, you add a ManageSieve service to the dispatcher, create a channel definition, and set optional channel options. Note that the ManageSieve channel does not act as either a source or destination channel in the conventional sense in the MTA. Instead, it is used to store various ManageSieve configuration settings. In particular, the channel is used to specify where Sieves are stored. The `filter` channel option specifies where the active Sieve is stored while the `destinationfilter` channel option specifies where the Sieve collection is stored.

Configuring the ManageSieve Service

To configure the ManageSieve Service:

1. Edit the `dispatcher.cnf file` and add the following lines if you are running in a legacy configuration.

   ```
   [SERVICE=MANAGESIEVE]
   PORT=4190
   IMAGE=IMTA_BIN:managesieve
   LOGFILE=IMTA_LOG:managesieve_server.log
   STACKSIZE=204800
   PARAMETER=CHANNEL=managesieve
   ```

   If you have a Unified Configuration, run the following `msconfig` commands.

```
msconfig
set dispatcher.service:MANAGESIEVE.tcp_ports 4190
set dispatcher.service:MANAGESIEVE.parameter "CHANNEL=managesieve"
set dispatcher.service:MANAGESIEVE.stacksize 204800
set dispatcher.service:MANAGESIEVE.image "IMTA_BIN:managesieve"
set dispatcher.service:MANAGESIEVE.logfilename
"IMTA_LOG:managesieve_server.log"
write
```

2. Create a channel definition specifying Sieve storage in LDAP by editing the `imta.cnf` configuration file in a legacy configuration and adding the following channel definition.

```
managesieve smtp_crlf filter ldap:///$A?mailSieveRuleSource \
destinationfilter ldap:///$A?mailSieveCollection maytlsserver
managesieve-daemon
```

If you have a Unified Configuration, run the following commands.

```
msconfig
set channel:managesieve.official_host_name managesieve-daemon
set channel:managesieve.smtp_crlf
set channel:managesieve.filter "ldap:///$A?mailSieveRuleSource"
set channel:managesieve.destinationfilter
"ldap:///$A?mailSieveCollection"
set channel:managesieve.maytlsserver
set channel:managesieve.scriptlimit 32
set channel:managesieve.sievelimit 1048576
set channel:managesieve.sizelimit 4194304
write
```

Note
The values of the last three channel options shown are the defaults. We include these settings to point out the limits imposed by the ManageSieve server.

The specification of LDAP URLs in the channel definitions indicates that scripts are to be stored in LDAP. The $A metacharacter substitutes the DN of the user's LDAP entry that was determined when the user authenticated to the ManageSieve server. Note the specification of the `maytlsserver` channel option. Although TLS is negotiated in the ManageSieve protocol and is therefore optional, most existing ManageSieve clients require it and will not work without TLS being enabled.

3. Set optional ManageSeive-channel-specific options. If you want to configure a custom implementation string in a legacy configuration, for example, add the following lines to a `managesieve_options` file.

```
CUSTOM_BANNER_STRING=Name
CUSTOM_VERSION_STRING=Version
```

To set these options in a Unified Configuration using `msconfig`, run the following commands.

```
msconfig
set channel:managesieve.options.custom_banner_string Name
set channel:managesieve.options.custom_version_string Version
write
```

A recipe, `managesieve.rcp`, is provided to set up a ManageSieve server in a Unified Configuration.

Supported Channel Options

The following table shows the channel options supported by the ManageSieve server and their descriptions:

Channel Option	Description
`filter` *`url`*	A URL specifying where the active script is stored. Either a `file:` or LDAP URL can be used. If an LDAP URL is used it must specify exactly one LDAP attribute. The following substitutions may be useful in constructing the URL: `$A` - The DN of the user's LDAP entry determined by the authentication process. `$M` - The user's UID determined by the authentication process. `$?`*whatever*`?` - The store's hashdir algorithm is applied to *whatever* to produce a directory path.
`destinationfilter` *`url`*	A URL specifying where the user's script collection is stored. Either a `file:` or LDAP URL can be used. If an LDAP URL is used it must specify exactly one LDAP attribute. The same set of substitutions provided for the `filter` option are available.
`scriptlimit` *integer*	Maximum number of scripts allowed in a single user's collection. The default is 32.
`sievelimit` *integer*	The maximum size, in octets, of a single script. Note that this limit is imposed at the protocol, not processing, level. The default is 1048576.
`sizelimit` *integer*	The maximum combined size, in octets, of a user's scripts. The default is 4194304.
`notlsserver`, `maytlsserver`, `musttlsserver`	Specifies whether TLS cannot be negotiated, may be negotiated, or must be negotiated, respectively. `notlsserver` is the default, but note that many ManageSieve clients require the availability and use of TLS.
`disconnectbadauthlimit` *`integer`*	Limit on the number of failed authentication attempts that can be done before the server disconnects.
`disconnectbadcommandlimit` *`integer`*	Limit on the number of bad commands the server will allow before disconnecting.
`disconnectcommandlimit` *`integer`*	Limit on the number of commands the server will allow before disconnecting.
`slave_debug`, `noslave_debug`	Controls whether or not the server produces debug output. `noslave_debug` is the default. Note that server logging does not normally include authentication information. Authentication information will be included if the `AUTH_DEBUG` ManageSieve-channel-specific option is set (see below).
`smtp_crlf`, `smtp_cr`, `smtp_lf`, `smtp_crorlf`	These options do not imply the use of the SMTP protocol. They are simply used to specify the line termination that the server requires for commands. `smtp_crlf` is the default and, given the use of counted literals in the protocol, should not be changed except under truly extraordinary circumstances.
`ident*`	Controls the sort of checks done on the client IP address. This information is only used for logging purposes in ManageSieve.
`nameservers` *`ip4-addrss-list`*	Controls what nameservers are used by any check requested by the `ident*` options. The default is to use the system default nameservers.

Supported ManageSieve-Channel-Specific Options

The following table shows the ManageSieve-channel-specific options supported by the ManageSieve server and their descriptions:

ManageSieve-Channel-Specific Option	Description
`CUSTOM_BANNER_STRING` *string*	Text to include in the `IMPLEMENTATION` capability response. The default is the Messaging Server product name.
`CUSTOM_VERSION_STRING` *string*	Text to include in the `IMPLEMENTATION` capability response. The default is the Messaging Server product version and build date.
`LOG_CONNECTION` *integer*	Same semantics as the `LOG_CONNECTION` TCPIP-channel-specific option.
`IGNORE_BAD_CERT` *integer*	Same semantics as the `IGNORE_BAD_CERT` TCPIP-channel-specific option.
`COMMAND_TIMEOUT` *integer*	The amount of time in seconds that the ManageSieve server will wait for a command. The default is 300 seconds (5 minutes).
`STATUS_TIMEOUT` *integer*	The amount of time in seconds that the ManageSieve server will wait for a status response to be sent. The default is 60 seconds.
`TRACE_LEVEL` *integer*	Enables protocol-level tracing of all protocol exchanges just above the TLS layer. Note that this will include authentication exchanges. The default value is 0.
`AUTH_DEBUG` *string*	Set of space-separated keys to pass to the authentication subsystem to enable various debug settings. See the HULA documentation for details about what keys are available. Note that setting `AUTH_DEBUG` and the `slave_debug` channel option (see above) will cause authentication command and response information to be included in the debug logs. The default is for no keys to be set.
`MAX_THREADS` *integer*	Same semantics as the `MAX_SERVER_THREADS` TCPIP-channel-specific option.

ManageSieve Server Use of PORT_ACCESS Mapping

The dispatcher checks connections to the ManageSieve server with the PORT_ACCESS mapping just as it does with any other connection it handles. The following table shows the server-specific PORT_ACCESS metacharacters that are relevant to ManageSieve and a description of how they work.

PORT_ACCESS Metacharacter	Description
$U	$U is the equivalent of setting the slave_debug channel option, except it only affects the current connection.
$G	Sets `TRACE_LEVEL` ManageSieve-channel-specific option value to 2 for this connection only.

Basic Command/Response Test Script

The following output shows a basic command/response test script.

```
"IMPLEMENTATION" "YoyoDyne file test 0.00000000001"
"VERSION" "1.0"
"MAXREDIRECTS" "32"
"NOTIFY" "mailto"
"SIEVE" "copy date editheader encoded-character envelope-auth envelope
envelope-dsn environment ereject extracttext foreverypart ihave index
imap4flags mime reject relational redirect-dsn subaddress spamtest
spamtestplus variables virustest body extlists fileinto notify enotify
regex vacation vacation-seconds"
"STARTTLS"
"SASL" "PLAIN"
OK
noop
OK "NOOP Completed"
noop "doof"
OK (TAG {4}
doof) "Done"
authenticate "plain" "xxxxxxxxxxxx"
OK "Authentication successful"
listscripts
OK
putscript "discard" "discard;"
OK "Sieve added"
listscripts
"discard"
OK
getscript "discard"
{8}
discard;
OK
putscript "keep" "keep;"
OK "Sieve added"
listscripts
"keep"
"discard"
OK
setactive "discard"
OK "Sieve activated"
listscripts
"keep"
"discard" ACTIVE
OK
setactive "keep"
OK "Sieve activated"
listscripts
"keep" ACTIVE
"discard"
OK
setactive "discard"
OK "Sieve activated"
listscripts
"keep"
"discard" ACTIVE
OK
```

```
setactive ""
OK "Active Sieve deactivated"
listscripts
"keep"
"discard"
OK
deletescript "keep"
OK "Sieve deleted"
deletescript "discard"
OK "Sieve deleted"
listscripts
OK
putscript "test" "refuse \"this is a test\";"
NO {72}
Refuse not listed in require clause prior to use [ "this is a test" ; ^]
putscript "test" "require \"refuse\"; refuse \"this is a test\";"
OK "Sieve added"
getscript "test"
{42}
require "refuse"; refuse "this is a test";
OK
renamescript "test" "test2"
OK "Sieve renamed"
listscripts
"test2"
OK
getscript "test2"
{42}
require "refuse"; refuse "this is a test";
OK
renamescript "test3" "test"
NO (NONEXISTENT) "Source Sieve does not exist"
putscript "discard" "discard;"
OK "Sieve added"
renamescript "test2" "discard"
NO (ALREADYEXISTS) "Destination Sieve already exists"
putscript "doof" ""
NO "Quoted string too short"
deletescript "discard"
OK "Sieve deleted"
deletescript "test2"
OK "Sieve deleted"
havespace "doof" 1234
OK
havespace "doof" 12345678
NO (QUOTA/MAXSIZE) "Maximum Sieve size exceeded"
havespace "doof" 123456789
NO (QUOTA/MAXSIZE) "Maximum Sieve size exceeded"
havespace "doof" 1234567890
NO "Number is too large"
```

```
logout
BYE "Disconnecting"
```

Chapter 47. Message Store Database Replication

Message Store Database Replication

Topics:

Overview of Message Store Database Replication

Bekerley Database provides support for building high availability (HA) applications based on replication. Message store database replication uses Berkeley Database HA facilities and low cost NFS storage devices to build an HA message store.

The `mboxlist` database is a transactional database. Database changes are written to the transaction logs. You can replicate the database by transporting the transaction log records from one site to another. Berkeley Database HA architecture supports single writer (master) and multiple reader replication. You must perform all database updates on the master. Replicas are available for read only activity. When the master fails, an election takes place, and one of the replicas will take over as master.

A message store replication group consists of one or more message store nodes. The message store nodes typically run on different physical hosts. The `mboxlist` database is replicated on every node. You can store the database locally. You store the mailbox partitions on remote storage devices running NFS servers. The NFS file systems are always mounted on all of the nodes in a replication group. You can configure the message store nodes with one or more remote hosts. If remote hosts are configured, the message store contacts a remote host to retrieve the replication group data on start up. If a master has not been established in a group, an election is called. A priority value is assigned to a node. When an election is held, the node with the most up-to-date log record and the highest priority becomes the new master. A node with priority 0 cannot be elected.

When the master fails, the replicas will automatically hold an election to select a new master. You are responsible for monitoring the master and redirecting traffic to the new master. The message store replicas run in read only mode. Any attempt to modify a mailbox on a replica returns an error. You can perform read only operations such as `mboxutil -l` and `imsbackup` on the replicas. You can install Multiple message store nodes on the same host in different zones. Each message store node must have a unique hostname and replication port number.

Berkeley Database maintains an internal database to keep track of the replication group data. Starting the first node in a replication group for the first time initializes the database. To start up the first node the first time, run `start-msg -m`. Database transaction commits blocks until it either receives enough acknowledgments, or the acknowledgment timeout expires. You can configure the acknowledgment policy and timeout.

A two-site replication group is particularly vulnerable to duplicate masters when there is a disruption to communications between the sites. Two-site replication is disabled by default. You can enable it with the `store.dbreplicate.twosites` option. When this option is disabled, the message store cannot take over as master if the original master fails in a replication group with only two sites. In the event this happens, the message store cluster will be unavailable for write access.

Configuration Options

Topics:

`msconfig` and `configutil` Configuration Options

The following table shows the configuration options (the same in both legacy and Unified Configuration), their descriptions, data types, and defaults.

Option	Description	Data Type	Default
`store.dbreplicate.enable`	Enables the `mboxlist` database replication feature.	boolean	false
`store.dbreplicate.port`	Specifies the `mboxlist` database replication port number.	integer	55000
`store.dbreplicate.dbremotehost`	Specifies a space-separated list of remote hosts in the replication group.	`host[:port][ host:[port]]...`	null
`store.dbreplicate.dbpriority`	Specifies the replication site priority in group elections. A special value of 0 indicates that this site cannot be a replication group master.	integer	100
`store.dbreplicate.ackpolicy`	Controls the replication manager transaction commit acknowledgment policy. All nodes in a cluster must have the same policy.	0=none,1=one,2=one peer,3=quorum,4=all peer,5=all available,6=all clients	3
`store.dbreplicate.acktimeout`	Specifies the amount of time, in seconds, the replication manager waits to collect enough acknowledgments from replication group clients before giving up and returning a failure indication.	number of seconds	1 second
`store.dbreplicate.twosites`	Enables a two-site replication group.	boolean	false
`store.dbreplicate.queuemax`	Specifies the replication manager incoming queue size limit.	integer	104857600

Command-Line Utilities

- The `imcheck subsystem` option prints the database replication statistics. For additional information see the discussion about `imcheck` in the Messaging Server Reference.
  To print the database replication statistics, run the following command:

  ```
  imcheck -s rep
  ```

- The `start-msg -m` option starts the message store as a replication master. For additional information see the disussion about `start-msg` in the Messaging Server Reference.
  To start the message store as a replication master, run the following command:

  ```
  start-msg -m
  ```

- The `stored -d` *site* option removes a replication site from the replication database. For additional information, see the discussion about `stored` in the Messaging Server Reference. To delete an `mboxlist` replication site called `grumpy` from the cluster, run the following command:

```
stored -d grumpy
```

Configuring Message Store Database Replication

The following configuration examples show you how to configure message store database replication.

- To Configure a Three Node Cluster for HA
- To Configure a Two Node Cluster to Offload Read Access:

To Configure a Three Node Cluster for HA

The following example shows how to configure a cluster with three electable nodes (host1, host2, and host3 at example.com). The message store partition is on a shared storage mounted at /zfssa/primary.

1. Run the following `msconfig` commands on host1.example.com if you are running a Unified Configuration:

```
set store.dbreplicate.enable 1
set store.dbreplicate.dbremotehost "host2.example.com
host3.example.com"
set partition:primary.path /zfsaa/primary
set schedule.task:snapshot.crontab -d
set schedule.task:snapshotverify.crontab -d
write
start-msg -m
```

   or the following `configutil` commands if you are running a legacy configuration:

```
configutil -o store.dbreplicate.enable -v 1
configutil -o store.dbreplicate.dbremotehost -v 'host2.example.com
host3.example.com'
configutil -o store.partition.primary.path -v /zfssa/primary
configutil -o local.schedule.snapshot -d
configutil -o local.schedule.snapshotverify -d
start-msg -m
```

2. Run the following `msconfig` commands on host2.example.com if you are running a Unified Configuration:

```
set store.dbreplicate.enable 1
set store.dbreplicate.dbremotehost "host1.example.com
host3.example.com"
set partition:primary.path /zfsaa/primary
set schedule.task:snapshot.crontab -d
set schedule.task:snapshotverify.crontab -d
write
start-msg
```

or the following `configutil` commands if you are running a legacy configuration:

```
configutil -o store.dbreplicate.enable -v 1
configutil -o store.dbreplicate.dbremotehost -v 'host1.example.com
host3.example.com'
configutil -o store.partition.primary.path -v /zfssa/primary
configutil -o local.schedule.snapshot -d
configutil -o local.schedule.snapshotverify -d
start-msg
```

3. Run the following `msconfig` commands on host3.example.com if you are running a Unified Configuration:

```
set store.dbreplicate.enable 1
set store.dbreplicate.dbremotehost "host1.example.com
host2.example.com"
set partition:primary.path /zfsaa/primary
set schedule.task:snapshot.crontab -d
set schedule.task:snapshotverify.crontab -d
write
start-msg
```

   or the following `configutil` commands if you are running a legacy configuration:

```
configutil -o store.dbreplicate.enable -v 1
configutil -o store.dbreplicate.dbremotehost -v 'host1.example.com
host2.example.com'
configutil -o store.partition.primary.path -v /zfssa/primary
configutil -o local.schedule.snapshot -d
configutil -o local.schedule.snapshotverify -d
start-msg
```

To Configure a Two Node Cluster to Offload Read Access:

The following example shows how to configure a cluster with one master and one replica. The message store partition is on a shared storage mounted at /zfssa/primary. The replica is read-only. It is not electable.

1. Run the following `msconfig` commands on master.example.com if you are running a Unified Configuration:

```
set store.dbreplicate.enable 1
set store.dbreplicate.ackpolicy 5
set partition:primary.path /zfsaa/primary
write
start-msg -m
```

   or the following `configutil` commands if you are running a legacy configuration:

```
configutil -o store.dbreplicate.enable -v 1
configutil -o store.dbreplicate.ackpolicy -v 5
configutil -o store.partition.primary.path -v /zfssa/primary
start-msg -m
```

2. Run the following `msconfig` commands on replica.example.com if you are running a Unified Configuration:

```
set store.dbreplicate.enable 1
set store.dbreplicate.ackpolicy 5
set store.dbreplicate.dbpriority 0
set store.dbreplicate.dbremotehost "master.example.com"
set partition:primary.path /zfsaa/primary
set schedule.task:snapshot.crontab -d
set schedule.task:snapshotverify.crontab -d
write
start-msg
imsbackup -r -f big /
```

or the following `configutil` commands if you are running a legacy configuration:

```
configutil -o store.dbreplicate.enable -v 1
configutil -o store.dbreplicate.ackpolicy -v 5
configutil -o store.dbreplicate.dbpriority -v 0
configutil -o store.dbreplicate.dbremotehost -v
'master.example.com'
configutil -o store.partition.primary.path -v /zfssa/primary
configutil -o local.schedule.snapshot -d
configutil -o local.schedule.snapshotverify -d
start-msg
imsbackup -r -f big /
```

Chapter 48. Message Store Manual Failover

Message Store Manual Failover

This chapter describes Messaging Server's Message Store manual failover feature and its configuration. It contains the following sections.

Basic Requirements

In order to use Message Store manual failover, the following is necessary.

- Messaging Server must be running in Unified Configuration mode.
- Messaging Server must be deployed using LMTP.

Overview of Message Store Manual Failover

The Message Store manual failover feature is useful for customers who already have 24/7 operators in their machine room but do not want the extra deployment and configuration expense of Sun Cluster and do not want the extra cost and vendor interaction of Veritas Cluster.

The basic model is that all Messaging Server hosts in the deployment need to be manually configured with an ordered list of hostnames for each mailStore. Each hostname corresponds to a separate product installation, but in a given mailStore list all the hosts must use a shared disk (e.g., NFS, filer) for the product data, but have a configuration that is largely identical except for the "base.hostname" setting. The first host in the list is the primary host for that mailStore. The primary host is running and the secondary hosts are not running (on standby).

When the primary host fails, the operator needs to make sure it has ceased to modify the shared disk, then start up services on the secondary host. The MMP, LMTP client, and imapd shared folder support will now automatically use the secondary host for requests related to the primary mailHost; there is no need to refresh or restart these services for this to happen.

Note that a standby hostname does not mean it is necessary to have unused hardware. If multiple IP addresses are used on the same server, then it can support multiple installations. However, if the primary host is dedicated and the secondary host is shared for a mailStore, then the service response time will be reduced when manual failover happens. Sites need to consider how much spare capacity is needed to service users when there is a hardware outage.

To enable safety during rolling version upgrades where a node is deliberately shutdown during upgrade, we recommend having at least 3 hostnames associated with each mailHost.

Configuring Message Store Manual Failover

For this example, we assume the following hosts in a deployment:

LDAP mailStore "store1.example.com"
store1a.example.com primary
store1b.example.com

store1c.example.com
LDAP mailStore "store2.example.com"
store2a.example.com primary
store2b.example.com
store2c.example.com
mta.example.com - an MTA configured to use LMTP
mmp.example.com - an MMP

We recommend that customers use Unified Configuration recipes to set up manual failover to avoid typographical errors when configuring multiple machines by running a recipe similar to the following on all machines in the deployment.

```
set_option("proxy:store1\\.example\\.com.storehostlist",
"store1a.example.com store1b.example.com store1c.example.com");
set_option("proxy:store2\\.example\\.com.storehostlist",
"store2a.example.com store2b.example.com store2c.example.com");
```

Save the recipe above to a plain text file, for example, `manual-failover.rcp`

Some extra configuration need be added to LMTP server and all the clients communicating to LMTP server.

To Configure the LMTP Server

1. Add the `manual-failover.rcp` file configuration to the existing LMTP server configuration. Or, if you prefer, there is also a sample recipe `LMTPBackendFailover.rcp` that configures a backend LMTP server for use with failover. If you wish to use this, you must copy the recipe script and manually add in to your LMTP client IP addresses and mailstore proxy information. This is available in the Messaging Server installed location *MessagingServer_Home* `/lib/recipes/LMTPBackendFailover.rcp`.
2. Run the recipe script on all back-end machines in the deployment by executing the following the command.

   ```
   msconfig run manual-failover.rcp OR LMTPBackendFailover.rcp
   ```

3. If you are running a compiled a configuration, recompile by running:

   ```
   imsimta cnbuild
   ```

4. Start the Messaging Server by running:

   ```
   start-msg
   ```

To Configure the Client

1. Add the `manual-failover.rcp` file configuration to all clients' configurations. For the LMTP client, you must add some extra configuration.
2. Run the recipe script on all client machines (MMP, LMTP client, and do on) in the deployment by executing the following command:

   ```
   msconfig run manual-failover.rcp
   ```

3. For the LMTP client only, you must set the `affinity list` channel option on the LMTP client

channel:

```
msconfig set channel:tcp_lmtpcs.affinitylist
imsimta cnbuild
stop-msg
start-msg
```

4. Stop the Messaging Server Client by running:

```
stop-msg mmp or mta
```

5. Start the Messaging Server Client by running:

```
start-msg mmp or mta
```

Chapter 49. Message Tracking and Recall

Message Tracking and Recall

Messaging Server includes a general message tracking and recall facility that conforms to RFCs 3885-3887. This includes support for the SMTP Extension for Message Tracking, Message Tracking Query Protocol (MTQP), and the Submission Future Release Extension (RFC 4865). The client provides tracking ID and a secret and future release delay during which the message can be recalled from the mail queue.

The recall aspect is implemented as a tracking extension. Additionally, some tracking extensions have been implemented to make it possible for a tracking client to track and recall messages submitted by a non-tracking client.

The following table shows the MTA and channel options used for and relevant to message tracking and recall.

Option	Description
`tracking_mode` *integer*	The `tracking_mode` MTA option controls how message tracking information is stored by the MTA. The default value of 0 disables storage of tracking information. Setting `tracking_mode` to 1 enables storage using a `memcached` server shared across the deployment. The use of other values is currently restricted. Note that the `memcache` protocol is implemented by many database systems. Any implementation that supports `memcached` semantics can be used as an alternative to `memcached`. Default 0.
`tracking_retries` *integer*	Tracking information updates can fail because of simultaneous access attempts to the underlying database. If this happens the update can be retried. The `tracking_retries` MTA option specifies how many times to retry the update. Default 5.
`tracking_retry_delay` *integer*	The `tracking_retry_delay` MTA option specifies the amount of time to delay between retry attempts in units of centiseconds. Default 10.
`tracking_debug` *integer*	The `tacking_debug` MTA option is used to enable debug output from the MTA's tracking subsystem. The default value of 0 disables debug output. Positive values enable it. The larger the value, the more output that is produced. Default 0.
`log_tracking` *integer*	The `log_tracking` MTA option controls logging of message tracking and recall information. Bit 0 (value 1), if set, includes the tracking ID and the current tracking in transaction log entries. The two appear as a single field. The ID appears first and is separated from the timeout value by a colon. Tracking information appears immediately after the envelope ID and before the deferred delivery time. An `mt` attribute is used in the XML log format. If bit 1 (value 2) is set in the log_tracking MTA option, then the tracking ID and timeout, again separated by a colon, appear in the `LOG_ACTION` mapping table probe, again immediately after the envelope ID and before the deferred delivery time. Default 0.

log_futurerelease	The log_futurerelease MTA option logs the values associated with the FUTURERELEASE SMTP extension. Setting bit 0 (value 1) causes the future release value, expressed as an offset in seconds from the current time, to be logged immediately after the mailbox UID is logged. An fr attribute is used in the XML log format. If bit 1 (value 2) is set in the log_futurerelease MTA option, then this information appears in the LOG_ACTION mapping table probe immediately after the mailbox UID.
notrackingclient, notrackingserver, trackingclient, trackingserver	These channel options control the availability and use of the Message Tracking SMTP extension specified in RFC 3885. Availability of the extension is enabled with the trackingserver source channel option. The notrackingserver channel option disables the availability of the extension, and is the default. The SMTP client's use of the extension is controlled by the trackingclient and notrackingclient channel options. The former enables use of the extension. The latter disables it and is the default. Note that the extension must be enabled on all internal SMTP clients and servers throughout a deployment in order for tracking and recall to work across that deployment.
trackingtimeoutdefault *integer*	The trackingtimeoutdefault option specifies the default timeout that's applied if the MTRK parameter is issued to the SMTP server without a timeout value. Default 259200 (3 days).
trackingtimeoutmax *integer*	The trackingtimeoutmax option specifies the maximum allowed timeout value in seconds. Any value greater than the maximum is silently lowered to the maximum. Default 1209600 (14 days).
trackingtimeoutmin *integer*	The trackingtimeoutmin option specifies the minimum allowed timeout value in seconds. Any value less than the minimum is silently raised to the minimum. Default 86400 (1 day).
trackinginternal, trackingrelayed, trackingdelivered	The handling of messages relayed internal to a deployment, including internal channel hops, needs to be distinguished from the case where messages leave the administrative domain for tracking and recall to work properly. However, the SMTP protocol is commonly used in both cases. Additionally, the case where a successful channel dequeue results in message delivery also needs to be distinguished from dequeues where this does not occur. Three channel options are provided to specify these semantics: trackinginternal, trackingrelay, and trackingdelivered. trackinginternal, the default, specifies that the message is being transferred internally. trackingrelayed specifies that the channel transfers messages to some external system. trackingdelivered specifies that the channel performs final delivery of the message.

`trackingmultiple`, `trackingsingle`, `trackingfirst`	RFC 3885 specifies how the Message Tracking SMTP Extension interacts with aliases and mailing lists. In particular, it says, "MTAs MUST NOT copy MTRK certifiers when a recipient is aliased, forwarded, or otherwise redirected and the redirection results in more than one recipient. However, an MTA MAY designate one of the multiple recipients as the "primary" recipient to which tracking requests shall be forwarded; other addresses MUST NOT receive tracking certifiers. MTAs MUST NOT forward MTRK certifiers when doing mailing list expansion." This arguably makes sense for tracking-only applications where presenting the results of a complex alias expansion process to the end user may be confusing. However, the situation with message recall is different. Users expect recall to work when feasible, including when alias expansion is involved. (Mailing lists are different; a mailing list effectively "owns" its messages once it expands, so recall past a mailing list expansion is inappropriate.) Accordingly, three source channel options are provided to control the MTA's behavior in this regard. `trackingmultiple`, the default, tells the MTA to pass tracking ID/timeout information to all recipients of an alias expansion. `trackingsingle` causes tracking ID/time information to pass through only when there is a single recipient. `trackingfirst` causes tracking information to pass through to the first alias expansion recipient. (Note that in the case of aliases stored in LDAP, the first recipient is unpredictable.)

`trackinggenerate` *n*	Message tracking and recall depends on the generation, attachment, and transfer of tracking identifiers. Such identifiers are normally generated and attached to messages by the submitting client. However, essentially no clients currently support the generation of such identifiers, making it impossible to write a separate tracking/recall client to deal with messages submitted by a non-tracking-enabled client. Additionally, a user who elects to use multiple clients, some tracking-enabled and some not, will end up with only a subset of their messages able to be tracked and recalled. Tracking identifiers can also provide, independent of their use for user tracking and recall, a stable identifier that ties MTA log entries across multiple systems together in ways that envelope IDs and Message-ID header fields do not and cannot. As such, automatic assignment of tracking identifiers to message on ingress as well as submission has real utility independent of user tracking and recall functions. The `trackinggenerate` source channel option addresses these needs. A single required integer parameter specifies the default tracking timeout. If set, a tracking identifier for the message is generated in one of two ways: • If authentication has been used and the user has a general recall secret associated with their LDAP entry, a per-message recall secret is generated by computing a SHA-1 hash of the concatentaiton of the content of Message-ID: header field , the Date: header field (if present), and the general recall secret. The tracking timeout is controlled by the `trackinggenerate` value. • If authentication was not used or no general recall secret is associated with the account, a tracking identifier is created by hashing a unique identifier with an MD4 hash. Note that the security of this process is controlled not by the randomness of the unique identifier or the use of MD4, but rather by the infeasilbility of computing X given T, where SHA1(X) = MD4(T).
`ldap_attr_auth_recall_secret`	The `ldap_attr_auth_recall_secret` MTA option specifies the name of the LDAP attribute where a user's general recall secret is stored. This option has no default.
`memcache_host`	Specifies the name of the host running either `memcached` or a `memcached`-compatible server where tracking and recall information will be stored. Note that each tracked message creates a single entry with the tracking ID as the key. The size of the entries is highly dependent on the number of recipients. Messages with large numbers of recipients can result in very large entries. This option has no default.
`memcache_port`	Specifies the `memcached` port on the host specified by `memcache_host`. Default 11211

Chapter 50. On Demand Mail Relay

On Demand Mail Relay

Support for On Demand Mail Relay as specified in RFC 2645 has been completed. This support piggybacks off existing support for SMTP `TURN` and doesn't involve any new options. Specifically, the `turn_in` channel option now enables both `TURN` and `ATRN`.

Note that the optional parameter to the `ATRN` command is only allowed if the `ODMR` channel is marked `single_sys` or `single`. This is so messages sent to a particular email domain can be reliably retrieved without getting messages sent to other domains.

It is strongly recommended that each administrative domain that requires `ATRN` service be configured as a separate channel rather than relying on `ATRN`'s domain selection capabilities. Specifically, the recommended configuration process to provide relay on demand for a new administrative domain foo is as follows:

1. Create a new channel for the domain foo:

   ```
   tcp_foo mustsaslserver single_sys slave smtp turn_in
   tcp_auth-daemon
   ```

   Or in `msconfig`:

   ```
   set channel:tcp_foo.official_host_name tcp_foo-daemon
   set channel:tcp_foo.mustsaslserver
   set channel:tcp_foo.single_sys
   set channel:tcp_foo.slave
   set channel:tcp_foo.smtp
   set channel:tcp_foo.turn_in
   ```

   Note the presence of the `slave` channel option. This prevents the channel from trying to perform regular SMTP deliveries. This can be removed if such delivery attempts are desired.
   Also note the presence of `single_sys`. `multiple` can be used instead and will be more efficient if multiple email domains are involved and the `ATRN` command is always used without an argument.

2. Create rewrite rules for all email domains associated with the foo administrative domain, for example:

   ```
   foo.example.com $U%$D@tcp_foo-daemon
   foo.example.org $U%$D@tcp_foo-daemon
   ```

   Or in msconfig:

```
set rewrite.rule foo.example.com $U%$D@tcp_foo-daemon
set rewrite.rule foo.example.org $U%$D@tcp_foo-daemon
```

3. Create a regular email account for foo with whatever name and credentials are desired. The account should include the LDAP attribute `mailSubmitChannel` with the value `tcp_foo`.
4. (optional) Create a dispatcher configuration identical to the regular service on port 25 except that it listens on port 366, the On Demand Mail Relay port.
5. (optional) Add a rule to the `FROM_ACCESS` mapping to block all mail coming from the ODMR port unconditionally.
6. (optional) Add a rule to the `FROM_ACCESS` mapping to block all mail from the tcp_foo channel unconditionally. This will prevent the administrative account from being used to send authenticated mail.

At this point `ATRN` should be usable by connecting, authenticating as the newly created user, and issuing an ATRN.

Chapter 51. Priority Message Handling

Priority Message Handling

This chapter describes Messaging Server's support of the MT-PRIORITY SMTP extension defined in RFC 6710. MT-PRIORITY values are always in the range -9 to 9. Priority 0 is the default.

The support for this extension consists of the following parts:

- The mtprioritiesallowed and mtprioritiesrequired source channel options. Both accept either one- or two-integer arguments. Two-integer arguments specify the range. You can specify the arguments in any order. The following table shows the channel options and their descriptions.

Channel Option	Description
mtprioritiesallowed *int1 [int1]*	Specifies the range of MT-PRIORITY values that will accepted. MT-PRIORITY values outside this range will be adjusted up or down so they fall within the allowed range. If a single argument is given it specifies the highest priority value that will be accepted. The default if this option is not specified is for the MT-PRIORITY extension not to be offered and for MT-PRIORITY parameters not to be accepted.
mtprioritiesrequired *int1 [int2]*	Specifies the range of MT-PRIORITY that will be accepted for enqueue. If a single argument is given it specifies the lowest priority value that will be accepted. The message will be rejected if the specified MT-PRIORITY value or the default value of 0 falls outside the required range.

- An INCLUDE_MTPRIORITY MTA option. This is a bit-encoded option. The default value is 0. The following table shows the bits, corresponding values, and descriptions of the INCLUDE_MTPROPRITY MTA option.

Bit	Value	Description
0	1	Appends the MT-PRIORITY and expected message size as separate fields to the FROM_ACCESS mapping probe immediately after any INCLUDE_SPARES values.
1	2	Appends the MT-PRIORITY and expected message size as separate fields to the FORWARD mapping probe immediately after the conversion tag field.
2	4	Appends the MT-PRIORITY and expected message size as separate fields to the ORIG_SEND_ACCESS mapping probe immediately after any INCLUDE_SPARES values.
3	8	Appends the MT-PRIORITY and expected message size as separate fields to the SEND_ACCESS mapping probe immediately after any INCLUDE_SPARES values.
4	16	Appends the MT-PRIORITY and expected message size as separate fields to the ORIG_MAIL_ACCESS mapping probe immediately after any INCLUDE_SPARES values.
5	32	Appends the MT-PRIORITY and expected message size as separate fields to the MAIL_ACCESS mapping probe immediately after any INCLUDE_SPARES values.
6	64	Appends the MT-PRIORITY and actual message size values in the form: ;MT-PRIORITY=<value>;BLOCKS=<value> to the conversion mapping probe immediately after any tag= clause.
7	128	Append the MT-PRIORITY and expected message size as separate fields to any domain catchall mapping probe immediately after the conversion tag.

The expected message size is the size of the queued message entry for internal channels. It is the value given by the SMTP SIZE extension for incoming SMTP channels. The size is given in MTA blocks.

- A LOG_MTPRIORITY MTA option. This is a bit-encoded option. The default is 0. The following table shows the bits, corresponding values, and descriptions of the LOG_MTPRIORITY MTA option. An mp element is used in the new XML log format to log the priority information. For more information about the XML log format, see the discussion on managing logging in the Messaging Server Administration Guide.

Bit	Value	Description
0	1	Enables logging of the MT-PRIORITY associated with each transaction. The MT-PRIORITY appears immediately after the message sensitivity and before the header-based priority in each log entry.
1	2	Message transfer priority appears in the LOG_ACTION mapping table probe, immediately after the sensitivity field and before the header-based priority field.

- Rewrite rule metacharacters that test both the current MT-PRIORITY value and the expected message size. For each metacharacter, *n* is a signed integer value. Note that the sign is required even when *n* is positive. The expected message size is the size of the queued message entry for internal channels. It is the value given by the SMTP SIZE extension for incoming SMTP channels. The size is given in MTA blocks.
  The following table shows the rewrite rule metacharacters and their descriptions.

Metacharacter	Description
$*n*<P	Rule succeeds only if *n* is less than the current MT-PRIORITY.
$*n*<=P	Rule succeeds only if *n* is less than or equal to the current MT-PRIORITY.
$*n*>P	Rule succeeds only if *n* is greater than the current MT-PRIORITY.
$*n*>=P	Rule succeeds only if *n* is greater than or equal to the current MT-PRIORITY.
$*n*=P	Rule succeeds only if *n* is equal to the current MT-PRIORITY.
$*n*<>P	Rule succeeds only if *n* is not equal to the current MT-PRIORITY.
$*n*<B	Rule succeeds only if *n* is less than the expected message size.
$*n*<=B	Rule succeeds only if *n* is less than or equal to the expected message size.
$*n*>B	Rule succeeds only if *n* is greater than the expected message size.
$*n*>=B	Rule succeeds only if *n* is greater than or equal to the expected message size.
$*n*=B	Rule succeeds only if *n* is equal to the expected message size.
$*n*<>B	Rule succeeds only if *n* is not equal to the expected message size.

- A -mtpriority switch added to imsimta test -rewrite, imsimta calc, and imsimta test -expression utilities. A single integer argument is required specifying the initial MT-PRIORITY value.

- A Sieve environment item, vnd.oracle.mt-priority. This item returns the current MT-PRIORITY value as a string.

- A nonstandard Sieve action, setmtpriority. This action accepts a single integer or string argument and sets the current MT-PRIORITY to the argument value. This action is only allowed in system-level Sieves and the argument must be in the -9 to 9 range of valid MT-PRIORITY values.

- A bit defined in the MESSAGE_SAVE_COPY_FLAGS MTA option. Bit 3 (value 8), if set, causes the MT-PRIORITY value for the current message to be included, delimited by vertical bars, immediately after the conversion tag.

- An MTPRIORITY_POLICY MTA option. This option is used to specify the priority handling policy

the MTA has been configured to support. This name is announced in the SMTP EHLO response on any channel where the `MT-PRIORITY` extension is enabled. The default is the empty string, meaning that no policy is announced.

Chapter 52. Implementing Greylisting by Using MeterMaid

Implementing Greylisting by Using MeterMaid

Topics:

What is Greylisting and How Does It Work?

Greylisting is a technique used by some MTAs as a way to reduce the number of undesirable spam messages they receive. Simply put, greylisting initially gives a temporary rejection to all incoming mail the first time it sees it, but then permits it upon subsequent attempts. It works by making the assumption that most spam is sent by spambots, PCs that have been compromised by a virus or trojan software, that act as mass-mailing clients. In order to send out as much spam as possible, these systems will connect to a mail server and attempt to deliver the spam to the recipient. If it should encounter a failure, it is extremely unlikely to retry delivery. Proper MTA clients will reschedule and reattempt delivery upon receiving a temporary failure, thus allowing the greylisting mail server a second chance to permit the message to be received.

Greylisting matches messages by using a combination of the source IP address, the envelope `FROM:` address, and the envelope `TO:` address. By using this triplet, greylisting works before the message body is sent during the SMTP transaction, making the temporary rejections happen in response to the `RCPT TO:` command. When the same triplet is presented to the mail server during a subsequent delivery attempt, the `RCPT TO:` command returns a successful response and the mail server will then accept the message for delivery.

In some setups, greylisting has been seen to reduce the number of spam messages received by 80-90%.

The downside to greylisting, however, is that it introduces an artificial delay to incoming mail from previously unknown triplets. The length of this delay varies depending on the originating MTA, but could range from thirty minutes to several hours. It is now expected by many that e-mail is nearly instantaneous, and greylisting can have a negative impact on customer's expectations.

Greylisting at a Glance: Example

Time	Action	SMTP Result	Explanation
9:45	Incoming SMTP transaction from `john@example.com` to `susan@siroe.com` (local user)	451 4.5.1 Temporary failure - retry later	This is the first time that Messaging Server has seen mail from `john@example.com` going to `susan@siroe.com` so Messaging Server responds with a temporary rejection.
9:47	Another transaction from `john@example.com` to `susan@siroe.com`	451 4.5.1 Temporary failure - retry later	Because this attempt happened within a short window after the first attempt, it is also temporarily rejected.
10:15	A bit later, the same transaction is retried	250 OK	Now that a subsequent attempt was made within the resubmit window, Messaging Server consider this combination of sender and recipient permitted.
10:20	Mail from `stephen@example.com` to `susan@siroe.com`	451 4.5.1 Temporary failure - retry later	This is a different sender, so it is handled independently from the previously permitted combination. This combination would be permitted after the block period has passed.
The next day	A new message from `john@example.com` to `susan@siroe.com`	250 OK	Since this combination is permitted, it remains valid for an extended period.

Different Implementations

Specific support for greylisting was introduced in Messaging Server 7 Update 4. MeterMaid was enhanced to support a greylisting table with additional related tuning options. It is possible, however, to implement a more basic form of greylisting using a throttle table which would work in earlier updates of Messaging Server 7.

Feature	Messaging Server 7 Update 3 or Earlier	Messaging Server 7 Update 4
Rejects previously unseen triplet	Yes	Yes
Continues rejecting for an initial blocking period (to block spambots that automatically retry within a very short period)	No	Yes
Once accepted, continues to accept messages from that triplet	Yes	Yes
Requires subsequent attempt within a specified period of time to register the triplet as permitted	No	Yes
Allows pre-registration of triplets based on outgoing mail, permitting wildcard source IP address	No	Yes
Expiration of existing triplets can be extended by recent use	No	Yes

Messaging Server 7 Update 4's support for greylisting has more functionality than using a throttle table.

Basic Greylisting Implementation

This section contains the following topics:

- Using Messaging Server 7 Update 4
- Using Messaging Server Versions 6.3 Through 7 Update 3

Using Messaging Server 7 Update 4

Setting up greylisting with Messaging Server 7 Update 4 is easier due to MeterMaid's built-in support for greylisting tables. Instead of calling the `throttle` routine in `check_metermaid.so`, you can use the `greylisting` routine which handles the appropriate return values for allowing Messaging Server to block transactions when the routine returns success.

First, the MeterMaid table definition:

```
metermaid.table.greylist.type = greylisting
metermaid.table.greylist.data_type = string
metermaid.table.greylist.max_entries = 50000
metermaid.table.greylist.options = nocase
metermaid.table.greylist.block_time = pt5m
metermaid.table.greylist.resubmit_time = pt4h
metermaid.table.greylist.inactivity_time = p7d
```

Note that the time formats can now be in ISO 8601 duration format. This feature was introduced in Messaging Server 7 Update 4.

This table defines a greylisting table with the following characteristics:

- All triplets are rejected during the first 5 minutes, even if multiple attempts occur.
- In order for a triplet to be recognized and permitted, a subsequent attempt must be made after that 5 minute window, but within 4 hours of the initial attempt.
- Once a triplet is permitted, it will remain as permitted in the table for 7 days after its last use.

The corresponding access control mapping table is simpler when using a greylisting table as no special handling by the mapping table is required:

```
ORIG_MAIL_ACCESS

! Check the source IP address, sender, and recipient in MeterMaid's
greylist table.
! If the call to greylisting() returns success, then Messaging Server
should return
!  a temporary rejection. If the call fails, then the greylisting check
has passed
! and other access control checks can continue.

  TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|*|l|* \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylist,$0|$1|$2]\
$N$X4.5.1|Temporary$ failure$ -$ retry$ later$E
```

(Note that the suggested form of the string being used in a greylisting table is *source-ip*|*env-sender*|*env-recipient*.)

After the MTA has received the `MAIL FROM:` and `RCPT TO:` SMTP commands, it will use the envelope addresses as well as the source IP address in a `greylisting` call to MeterMaid. Using the table configuration above, MeterMaid will determine whether or not this particular transaction should be

permitted. If the MTA should send a temporary rejection at this point, the call to `check_metermaid.so` will succeed, and the `$N` part of this entry will be returned indicating a rejection. The `$X4.5.1` flags this rejection as a temporary condition so that a `4xx` SMTP response will be given.

Using Messaging Server Versions 6.3 Through 7 Update 3

MeterMaid can implement a form of greylisting using a throttle table. In general, throttle tables are used to deny transactions that exceed certain limits. In this example, Messaging Server will use a throttle table to *permit* transactions that exceed a very low limit (usually 1, representing the number of temporary rejections the mail server will give before allowing receipt of the message). This means that when the call to the `throttle` routine in `check_metermaid.so` succeeds, Messaging Server will have seen this particular triplet before and can permit the transaction. This is the opposite of what is traditionally done with a throttle table.

For example, you could set up a basic MeterMaid greylisting table using this configuration:

```
metermaid.table.greylist.type = throttle
metermaid.table.greylist.data_type = string
metermaid.table.greylist.max_entries = 50000
metermaid.table.greylist.options = nocase
metermaid.table.greylist.quota = 1
metermaid.table.greylist.quota_time = 604800
```

These entries define a new throttle table for MeterMaid called greylist. By specifying `quota = 1`, Messaging Server can determine whether a triplet entry is being seen for the first time or if it has been seen before (and would then be over quota for the purposes of a throttle table). `quota_time = 604800` specifies that MeterMaid will keep track of these entries for a period of 7 days before forgetting about them. Note that this does mean that after 7 days, previous knowledge of an established entry will be lost and a sending mail client will once again get a temporary rejection.

Then, one would need to set up a new mapping table to handle the call to `check_metermaid.so` and hook that into the access control mapping table:

```
X-GREYLIST

! First, attempt to probe the greylist throttle table to see whether the
server
! has seen this particular entry before. If it has, then it can now
permit it.
! By returning $N from here, the calling mapping table will receive this
as
! a failure, and proceed down through the rest of the access checks.
!
! Then, if this is the first time the server has seen this entry, it
will
! return $Y, allowing the calling mapping table to continue with a
temporary
! rejection.
!
! The basic purpose of this mapping table is to invert the general use
of a
! MeterMaid throttle table.

  *     $C$[IMTA_LIB:check_metermaid.so,throttle,greylist,$0]$N$E
  *     $Y

ORIG_MAIL_ACCESS

! Check the GREYLIST mapping table for external connections to local
recipients
! to see whether they'll be permitted or receive a temporary failure
(expecting
! that they will retry again later, at which point the transaction would
be
! permitted).

  TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|*|l|*
$C$|X-GREYLIST;$0$|$1$|$2|\
$N$X4.5.1|Temporary$ failure$ -$ retry$ later$E
```

Here, in the `ORIG_MAIL_ACCESS` mapping table, incoming connections from external sites (such that the source channel is tcp_local) with local mail recipients (matching the l channel) are subject to the greylisting check. The `ORIG_MAIL_ACCESS` mapping table will then check the `X-GREYLIST` mapping table which does the actual MeterMaid check.

If the call to `check_metermaid.so` succeeds, that means that the entry in the greylist table has exceeded its throttle quota. In other words, MeterMaid has seen this entry before (and its hit count is now at least 2, which exceeds the previously defined `quota` value). When this occurs, you end the `X-GREYLIST` mapping table processing with `$N$E`. The calling line in the `ORIG_MAIL_ACCESS` mapping table considers this a failure and would then continue processing the remaining lines in the mapping table.

If, on the other hand, the call to `check_metermaid.so` fails, this means that the entry is new to the greylist table (its quota has not yet been exceeded since its hit count is 1 and `quota` is 1). The processing continues with the second `X-GREYLIST` table entry that ends with `$Y` indicating success to the caller. Since the call to the `X-GREYLIST` table succeeded, the rest of the line is processed, returning the `$N` flag to the SMTP server to send the specified rejection message. Note the `$X4.5.1` - this sets the SMTP response code to a 4xx value indicating a temporary failure to the remote mail client.

Enhancing Greylisting Functionality

However greylisting is configured, there are additional steps one can take to make further improvements to its functionality. Several possibilities are listed here. They can be used individually or combined together to make more powerful setups.

This section contains the following topics:

Preloading the Greylisting Table with Outbound Transactions

Since it is often the case that one can expect to receive mail from addresses to which the local users are already sending, it may be useful to preload such address combinations into the greylist table. Since the future source IP address is not known, MeterMaid supports a special address of * that will match any other supplied address.

To preload the address combinations, one needs to add an entry to the access control mapping table:

```
X-IS_INTERNAL_CHANNEL

  tcp_intranet         $Y
  tcp_submit           $Y
  tcp_auth             $Y
  *                    $N

ORIG_MAIL_ACCESS

! For mail that is coming from a local user and going to an external
recipient, we
! can save that user/recipient combination and store it into the
greylist table for
! future permission.

  TCP|$@*|$@*|$@*|$@*|SMTP$@*|MAIL|*|*|tcp_local|*
$C$|X-IS_INTERNAL_CHANNEL;$0|\
$[IMTA_LIB:check_metermaid.so,store,greylist,*|$2|$1,1]
```

(Note that the `store` routine requires a value although it is not used by a greylisting table. Any value may be specified here and is ignored by MeterMaid.)

This mapping table entry first checks to see whether the source channel is considered a channel used by our local users. If the channel is in the list provided by the `X-IS_INTERNAL_CHANNEL` mapping table, then processing continues with the call to the `store` routine of `check_metermaid.so` to store this new combination into the `greylist` table. The combination is stored so that it will match incoming messages from the current recipient going to the current sender, and these messages may come from any source IP address and be permitted.

Matching a Range of IP Addresses

A complication that can occur with greylisting is dealing with remote MTAs that use several different hosts to process deliveries. This can mean that one attempt may occur from 192.168.12.1, but a subsequent attempt may come from a different host like 192.168.12.5. Additional delays may be introduced until an attempt is repeated from a host that had tried it previously.

One way to help address this is to limit IP address matching to the first three octets, allowing more hosts to be considered to be the same source. This would match addresses coming from the same A.B.C.D/24 (class C) subnet. The mapping table setup would be very similar to the examples above, but with a change to the IP address wildcard matching.

```
ORIG_MAIL_ACCESS

! When checking the source IP address, only use the first three octets
in the string
! passed to MeterMaid.

  TCP|$@*|$@*|$D*.$D*.$D*.$@*|$@*|SMTP$@*|MAIL|tcp_local|*|l|* \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylist,$0.$1.$2|$3|$4]\
$N$X4.5.1|Temporary$ failure$ -$ retry$ later$E
```

Simplifying the Sender Address

Some sender addresses will be more complex than a simple `user@example.com` including such features as subaddresses or VERP (variable envelope return path) notation. It may be useful to help greylisting recognize the base form of the address using some basic canonicalization in order to keep track of the basic, simplified address form. This simplification can be done by using a nested mapping table call out to perform the canonicalization.

```
X-CORRESPONDENT

! Subsidiary mapping for removing any subaddress or VERP style material
from
! the local-part of an address.

  $_*$[+=\-]%*@*    $0@$3$Y
  *                 $0$Y

ORIG_MAIL_ACCESS

  TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|*|l|* \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylist,$0|$|X-CORRESPONDENT;$
failure$ -$ retry$ later$E
```

Here, the `X-CORRESPONDENT` table is used to reconstruct the sender address into the simpler `user@example.com` form. The result from this is then used in the call to the `greylisting` function.

Providing an Opt-In Mechanism

> **Note**
> This section requires Messaging Server 7 Update 2 or later for the necessary `INCLUDE_SPARES` option.

It may be desirable to allow users to choose whether to have MeterMaid perform greylisting on their incoming mail. This could be especially useful when considering some local mail recipients who may not wish to be subject to delays in receiving incoming mail from unknown senders, such as recipients like `sales` or `customer_service`. For these local users, one can set up additional LDAP attributes to be used in conjunction with the existing `ORIG_MAIL_ACCESS` mapping table processing.

For this example, let us assume that one creates a new LDAP attribute `mailUserGreyListOptIn` that will be set to `true` or `false`. Those users who have this attribute set to `true` will have their incoming mail checked with greylisting, while those who have it set for `false` will skip this check and receive their mail immediately.

In order to have the MTA look at this extra attribute, it must be configured into the `option.dat` configuration file.

```
LDAP_SPARE_5=mailUserGreyListOptIn
! Include LDAP_SPARE_5 in ORIG_MAIL_ACCESS probes by setting bit 22
(counting from 0)
! of INCLUDE_SPARES. Bit 22 has the value 4194304.
INCLUDE_SPARES=4194304
```

This will add the value of the user's `mailUserGreyListOptIn` attribute to the probe string used in the `ORIG_MAIL_ACCESS` mapping table.

```
ORIG_MAIL_ACCESS

! This example assumes INCLUDE_SPARES=4194304 is set, so that probes
corresponding
! to submissions from remote (tcp_local) senders to local recipients
have the form:
!
!
TCP|host-ip|host-port|source-ip|source-port|SMTP-app-info|MAIL|tcp_local| !
remote-sender-address|l|local-recipient-address|recipient-mailUserGreyListO
TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|$_*|l|$_*|true \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylist,$0|$1|$2]\
$N$X4.5.1|Temporary$ failure$ -$ retry$ later$E
```

The key difference in this mapping table entry is the addition of `|true` to the matching string. Since the `INCLUDE_SPARES` option will append the content of the `mailUserGreyListOptIn` attribute to the probe string, this mapping entry can match against only those where the recipient's `mailUserGreyListOptIn` attribute has been set to `true`, thus skipping others who may be opting out of greylisting.

Whitelisting Based on User's Addressbook

> **Note**
> This section requires Messaging Server 7 Update 2 or later for the necessary `INCLUDE_SPARES` option.

The goal of greylisting is to allow mail from remote senders to local recipients once they are known. In most cases, this happens when a transaction presents this combination on a subsequent delivery attempt. It is also possible to use the recipient's LDAP-based personal address book to check for the sender's address to determine whether to bypass greylisting for an already recognized address. In order

to do this, another LDAP attribute must be added to the ORIG_MAIL_ACCESS probe string by including these values into the option.dat file:

```
LDAP_SPARE_6=psroot
! Include LDAP_SPARE_6 in ORIG_MAIL_ACCESS probes by setting bit 23
(counting from 0)
! of INCLUDE_SPARES. Bit 23 has the value 8388608.
INCLUDE_SPARES=8388608
```

Furthermore, if appropriate, the MTA's LDAP_PAB_xyz options should be set to the proper values for accessing the PAB LDAP server. (However, the usual local.service.pab. settings in configutil are usually adequate, and usually do not need to be overridden for MTA purposes via the MTA-specific LDAP_PAB_xyz options.)

```
ORIG_MAIL_ACCESS

!
This example assumes INCLUDE_SPARES=4194304 is set, so that probes
corresponding
! to submissions from remote (tcp_local) senders to local recipients
have the form:
!
!
TCP|host-ip|host-port|source-ip|source-port|SMTP-app-info|MAIL|tcp_local| !
remote-sender-address|l|local-recipient-address|recipient-psroot

!
! Matches on this line mean that the sender was found in the recipient's
address book.
! "Whitelist" those addresses, bypassing the greylisting check.
!
  TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|$_*|l|$_*|* \
$C$]pabldap:///$3?piEmail1?sub?(|(piEmail1=$1)(piEmail2=$1)(piEmail3=$1))[$
Now, for all other senders, do the normal greylisting check.
!
  TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|$_*|l|$_*|* \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylist,$0|$1|$2]\
$N$X4.5.1|Temporary$ failure$ -$ retry$ later$E
```

This mapping table example shows an LDAP callout being done to check to see whether the sender is already known to the local recipient. This allows users to add their correspondents to their address book as a way to whitelist those entries, allowing those senders to bypass the greylisting when sending mail to these local recipients.

Combining Functionality: A Complex Example

It is possible to combine many of these elements together into a much more comprehensive setup. This example makes use of the preloading, opt-in, and address book whitelisting features together.

First, the two LDAP attributes must be available to the mapping table probe. They can be added with these options in option.dat:

```
LDAP_SPARE_5=mailUserGreyListOptIn
LDAP_SPARE_6=psroot
! Include LDAP_SPARE_5 and LDAP_SPARE_6 in ORIG_MAIL_ACCESS probes by
! setting bits 22 and 23 (counting from 0) of INCLUDE_SPARES; that is,
! INCLUDE_SPARES=12582912=4194304+8388608=(1<<22)+(1<<23)
INCLUDE_SPARES=12582912
```

Then, the `mappings` file excerpt below shows how the above elements may be combined.

```
! Subsidiary mapping for checking incoming port and channel against a
! list of "internal submission" channels.
!
! Probe format is
!    port.channel
!
X-INTERNAL-CHANNELS

  587.tcp_submit    $Y
  25.tcp_auth       $Y
  25.tcp_intranet   $Y

! Subsidiary mapping for removing any subaddress or VERP style
! material from the local-part of an address.
! This mapping also performs LDAP URL style quoting of the retained
! portion of the address.
!
X-CORRESPONDENT

   $_*$[+=\-]%*@*    $=$0@$3$_$Y
   *                 $=$0$_$Y

ORIG_MAIL_ACCESS

! This example assumes INCLUDE_SPARES=12582912 (or some superset of
bits) is
! set, so that probes corresponding to submissions from local senders to
! remote recipients have a form of:
!
!
TCP|host-ip|host-port|source-ip|source-port|SMTP-app-info|MAIL|source-chann
local-sender-address|tcp_local|remote-recipient-address|
! sender-mailUserGreyListOptIn|sender-psroot
!
! while probes corresponding to SMTP MAIL submissions from remote
! (tcp_local) senders to local recipients have the form:
!
!
TCP|host-ip|host-port|source-ip|source-port|SMTP-app-info|MAIL|tcp_local| !
remote-sender-address|l|local-recipient-address|
!  recipient-mailUserGreyListOptIn|recipient-psroot
!
! The overall logic includes pre-population of the "greylist" table at
(1)
```

```
! with *|remote-correspondent|local-user on outgoing messages from local
users
! who have opted-in to greylisting (have mailUserGreyListOptIn: true)
(0),
! and then checks of incoming messages to local users who want
greylisting (2)
! against:
!   (i) the local-user's PAB (3)
!   (ii) the pre-populated entries in the "greylist" table (4) or (5)
!   (iii) the "greylist" table tracking "recent" submission attempts
from
!         not-otherwise-known (not pre-populated due to local-user
sending
!         to them, nor recognized in local-user's PAB) remote senders
(4) or (5)
! Note that (ii) and (iii) are done by one probe to the greylist table,
as
! MeterMaid first performs the (ii) check automatically due to the
format of
! probe.  This probe the greylist table is either done at (4) (for IPv4
! source IPs) or at (5) (for IPv6 source IPs).
!
! For outgoing messages, from local users to remote correspondents,
! pre-populate the "greylist" table using the "store" entry point with
! *|simplified-quoted-remote-correspondent|local-user
! This is so that replies from that remote-correspondent (from whatever
! source-IP) to that local-user will be accepted.
! The subsidiary mapping table X-INTERNAL-CHANNELS is used to check
! (based on the host-port and source-channel) whether the message is
! one from a local user to a remote correspondent.  The subsidiary
! mapping table X-CORRESPONDENT is used to canonicalize the
! recipient-address.
! (0)
!
  TCP|$@*|*|$@*|$@*|SMTP$@*|MAIL|*|*|tcp_local|*|true|*   \
     $C$|X-INTERNAL-CHANNELS;$0.$1|PREPOPULATE|$|X-CORRESPONDENT;$3||$2
!
! If the message was indeed from a local user who wants grey-listing,
! then the above entry matched and reset the probe to now be:
! PREPOPULATE|quoted-simplified-remote-correspondent|local-user
! Then the entry below pre-populates the greylist table with an
! entry for
! *|quoted-simplified-remote-correspondent|local-user
! (1)
!
  PREPOPULATE|*|*     \
     $C$[IMTA_LIB:check_metermaid.so,store,greylist,*|$0|$1,1]$E
!
! For incoming submission attempts from remote correspondents (tcp_local
! submission attempts):
!   (2) Entry matches remote senders to recipients that
!       want grey-listing (mailUserGreyListOptIn: true).  Entry
constructs
!       a new GREYLIST... probe retaining relevant fields, namely:
!       GREYLIST|source-ip|quoted-simplified-sender|recipient|psroot
!       where the quoted-simplified-sender field is processed
```

```
(simplified
!       and LDAP quoted) using the subsidiary X-CORRESPONDENT mapping
table
!   (3) For the recipients who want grey-listing, the probe is now
!       GREYLIST|source-ip|quoted-simplified-sender|recipient|psroot
!       Look up the (simplified) sender address in the
!       recipients PAB. Accept submission if found, fall through
otherwise.
!   (4) If the sender address wasn't found at (3), fall-through and now
!       attempt a MeterMaid "greylist" table lookup.  The probe to this
!       "greylist" table will have the form:
!       source-IP-subnet|quoted-simplified-sender|recipient
!       This entry matches on IPv4 incoming source IPs, and ignores the
last
!       eight bits to give an IPv4 subnet.
!       Because the probe has the form A|B|C, and it is a probe to a
!       greylisting entrypoint, MeterMaid will automatically initially
attempt
!       a *|B|C probe, only bothering with the A|B|C probe if its
initial,
!       automatic probe fails.  Thus any pre-populated, generic source
IP
!       entry will match first, prior to MeterMaid attempting a lookup
of
!       the specific source IP subnet.  If the specific triad is
!       found in the "greylist" table as being due to be greylisted
(rejected
!       temporarily), then the probe "succeeds" -- set a new probe
string
!       FIRSTATTEMPT and continue so that (6) will match and the
greylist
!       response will be issued.
!       Otherwise, the MeterMaid probe "fails" -- as for the cases
!       where the probe matches a "good" (pre-populated, or resubmitted
!       after block_time) entry.
!   (5) The same as (4), but matching on IPv6 incoming source IPs,
ignoring
!       the last 64 bits to give an IPv6 subnet.
!   (6) Issue the temporary rejection when the greylist probe of (4) or
(5)
!       "succeeded".
!
! For remote senders (source channel tcp_local) to local recipients with
the
! LDAP_SPARE_5 attribute "true", reset the probe to the GREYLIST|...form
! (2)
!
  TCP|$@*|$@*|*|$@*|SMTP$@*|MAIL|tcp_local|$_*|l|$_*|true|*  \
$CGREYLIST|$0|$|X-CORRESPONDENT;$1||$2|$=$3$_
!
! If a recipient wants grey-listing, the probe has been rebuilt to be:
! GREYLIST|ip-source|simplified-sender-address|recipient-address|psroot
! where simplified-sender-address omits any subaddress/VERP-y sorts of
fluff
! and has had any LDAP URL required quoting applied, and where psroot
has also
```

```
! had any LDAP URL required quoting applied.
! So next check whether the simplified-sender-address can be found in
! the recipient-address user's PAB (found under psroot); if the sender
is
! found, then accept this message -- this sender is "known".
! (3)
!
  GREYLIST|*|*|*|*   \
$C$]pabldap:///$3?piEmail1?sub?(|(piEmail1=$1)(piEmail2=$1)(piEmail3=$1))[$
Otherwise, if the sender was not known to this recipient, then fall down
! to the subsequent entry which performs the MeterMaid grey-list check.
!
! Match on IPv4 addresses and probe the greylist table.
! If this sender matches an entry in the greylist table, whether
! pre-populated or due to a recent sending attempt, then let their
message in.
! If the greylist table probe says the sender needs greylisting,
! (that is, never seen before, or seen before but only within
block_time),
! then continue with the probe changed to "FIRSTATTEMPT" so that it'll
fall
! through and match the temporary rejection entry below at (6).
! Otherwise, if the sender was in the greylist table but after
block_time,
! then MeterMaid "fails" this probe, so the check ends; the table
processing
! "falls-through" and, if no other entry matches, the submission is
! permitted.
! (4)
!
  GREYLIST|$D*.$D*.$D*.$D*|*|*|*      \
$[IMTA_LIB:check_metermaid.so,\
greylisting,greylist,$0.$1.$2|$4|$5]$CFIRSTATTEMPT
!
! (5)
!
  GREYLIST|$H*:$H*:$H*:$H*:$@H*:$@H*:$@H*:$@H*|*|*|*      \
$[IMTA_LIB:check_metermaid.so,\
greylisting,greylist,$0:$1:$2:$3|$4|$5]$CFIRSTATTEMPT
!
! Must be a "new" sending attempt -- give it a temporary rejection
! (6)
```

```
!
  FIRSTATTEMPT     $N$X4.5.1|Temporary$ failure$ -$ retry$ later
```

Mapping Table Notes

The mapping tables above use several features that may be unfamiliar to casual users of the MTA's mapping table, including these:

Strings	Explanation
`$@`	Disables saving the following wildcard match that will not be needed for right-hand side substitutions. This permits sufficient saved wildcards (of which there can be at most ten) to be available for matching fields of more interest.
`$_`	Specifies "non-greedy" (minimal) matching of the portion of the string; used for the local-part of the sender address prior to the first occurrence of a special character possibly indicating a subaddress or VERP address variation.
`$[+=\-]%`	Matches an occurrence of any one of the specified characters. Note that the hyphen character must be quoted with a backslash character to be interpreted as a literal hyphen character rather than indicating a character range.
`$D*`	Matches only decimal digits; used for parsing IP addresses.
`$H*`	Matches only hexadecimal digits; used for parsing IPv6 addresses.

Chapter 53. Using the iSchedule Channel to Handle iMIP Messages

Using the iSchedule Channel to Handle iMIP Messages

Messaging Server is capable of posting a calendar event received in an iMIP (iCalendar Message-Based Interoperability Protocol) message to Calendar Server by using the iSchedule protocol. This capability enables "internal" users to automatically process calendar invitations from "external" users. To enable this interoperability between calendaring systems, you configure a Messaging Server "iSchedule" channel to process the iMIP messages. For additional information, see the topic on enabling the iSchedule channel to handle iMIP messages in *Calendar Server System Administrator's Guide*.

> This capability was introduced in **Communications Suite 7 Update 4** (Messaging Server 7 Update 5 and Calendar Server 7 Update 3).

Topics:

Inviting Users on Internal and External Calendar Systems Background

Calendar Server meetings often have multiple invitees. These invitees can be both *internal* users, who reside on the same Calendar Server deployment, or *external* users, who reside either on a different Calendar Server deployment administered by a separate group, or on an outside calendaring system, such as Exchange, Google Calendar, and so on. For "internal" invitees, Calendar Server automatically adds the meeting request to their calendars (referred to as *implicit scheduling*) and also sends them email notification about the meeting request. All "external" invitees are sent an iMIP (iCalendar Message-Based Interoperability Protocol) email with the meeting request as an attachment. External invitees must manually process these messages to add the invite to their calendars.

Manually Accepting External Invitations

In Calendar Server 7 Update 2, meeting invitations from external organizers are sent to the user's mailbox. Mail clients, such as Outlook or Thunderbird, enable users to process these invitations and add the invitation to their calendar. How the invitation is added to the user's calendar depends on the specific mail client, but the invitation is not added until the user has manually read the email and accepted the meeting request. In Calendar Server 7 Update 2, automatically accepting external invitations is not possible.

Automatically Accepting External Invitations

Starting with Calendar Server 7 Update 3 (in conjunction with Messaging Server 7 Update 5), you can configure your Calendar Server deployment to automatically process invitations coming from external calendar systems. To users, handling an external invite then appears just like an internal invite.

This capability involves an intermediary in the form of Messaging Server. You configure the Messaging Server MTA to process the calendar invite email (which is an iMIP message), extract the pertinent calendar information, then use the iSchedule protocol to add the invite to the attendee's calendar database. As a consequence, external event invitations automatically appear in the user's calendar without the need for a manual intervention, even when using a "non-calendar" aware client.

Once you have configured your deployment accordingly, users have a choice on how to process invitations. Users can either accept the "external" meeting invite directly from their calendar client (either desktop or mobile iOS CalDAV clients) or they can still accept it from their email client. That is, CalDAV clients now receive iMIP messages in their scheduling-inbox, and are able to process them just like regular CalDAV-based invitations and replies. Because the invitation is already in the user's calendar, invitation replies and cancel are also merged automatically. Thus, based upon the user accepting or rejecting the request, the calendar client merely has to update the attendee status in the invitation. That status change also enables Calendar Server to send a response to the organizer indicating the disposition of the meeting request. As the response is sent directly by Calendar Server, it does not matter how the user accepted the invitation (whether from a calendar client on a mobile device, desktop, or from an email client). Finally, because of the addition of meta data to the email message, the Convergence (web-based) client is able to display a scheduling-specific form to users that enables them to accept, decline, or indicate a "maybe" to meeting invitations directly from their email without having to switch to the calendar client.

Message Server iMIP Configuration Overview

A Messaging Server MTA channel (an "iSchedule" channel) handles automatic processing of external calendar invites by:

1. Intercepting incoming emails containing an iMIP message
   An iMIP message has an iCalendar attachment of type 'text/calendar' with a `method=<action>` parameter in the 'Content-Type:' header.
2. Injecting the corresponding iTIP message into the regular calendar server workflow
3. Adding meta-data (email X- headers) to the iMIP email before delivering it to its recipients
4. Posting the invitation request to a calendar iSchedule URL

The Calendar Server then consumes the invitation from the iSchedule URL just as it would have done if an external calendar server had posted an invitation to one of its users. On the Calendar Server side, an iSchedule database, which is a separate table from the Calendar Server database, acts as the global inbox and outbox for external invites.

Administering the iMIP configuration involves:

- Enabling or disabling iMIP messaging processing
- Configuring the iSchedule service URL
- Configuring the criteria for messages to be selected for processing

You should configure the iSchedule channel on the message store systems if you are not using LMTP. If you are using LMTP, configure the iSchedule on the MTAs.

For more information on the iCalendar Message-Based Interoperability Protocol, see RFC 6047. For more information on iCalendar Transport-independent Interoperability Protocol (iTIP), see RFC 5546.

Configuring the iSchedule Channel for iMIP Messages in Unified Configuration

You can use a Messaging Server Unified Configuration recipe to automate the configuration process or you can manually perform the necessary configuration. After completing the configuration, you also need to verify the Calendar Server configuration, as described in Verifying the Calendar Server Configuration.

You do not need to perform any additional Convergence configuration for Convergence to automatically process invitations coming from external calendar systems. If you have configured Messaging Server and Calendar Server correctly, Convergence users see a UI form that they use to reply to external invites.

Topics in this section:

Using the iSchedule Recipe to Automate Configuring the iSchedule Channel in Unified Configuration

Unified Configuration provides a recipe language and some stock recipes to automate certain configuration tasks. To set up the iSchedule channel, you can use a recipe called `iSchedule.rcp`, which automatically sets up the channel definition, job controller configuration, channel options, Sieve rule, and CONVERSION mapping.

To use the `iSchedule.rcp` recipe:

1. Run the `msconfig` command with the recipe name.

```
/opt/sun/comms/messaging64/bin/msconfig run iSchedule.rcp
```

2. Respond to the prompts, for example:

```
HTTP URL for iSchedule server:
http://host1.example.com:8080/dav/ischedule/
Destination channel for messages to check (<RET> if no more):
ims-ms
```

   Use the iSchedule URL and destination channels based on your deployment.

> **Note**
> Be sure to add the the trailing forward slash (/) in the iSchedule URL, otherwise you will receive the error message "HTTP Error 401 Unauthorized."

3. If you are using a compiled configuration, recompile the configuration.

```
/opt/sun/comms/messaging64/bin/imsimta cnbuild
```

4. Restart Messaging Server.

```
/opt/sun/comms/messaging64/bin/stop-msg
/opt/sun/comms/messaging64/bin/start-msg
```

5. Verify the Calendar Server configuration.
   See Verifying the Calendar Server Configuration

Manually Configuring the iSchedule Channel in Unified Configuration

The high-level steps to manually configure the iSchedule channel by using the `msconfig` command involve:

- Adding the channel
- Configuring the conversion mapping
- Specifying messages to be processed by iSchedule

To manually configure the iSchedule Channel, job controller master command, `include_conversiontag` MTA option, and conversion mapping:

1. Use the `msconfig` command in interactive mode to configure the iSchedule channel, the job controller master command for the channel, the `include_conversiontag` MTA option if you want to have a `TAG=` clause included in your conversion mapping probes, and the conversion mapping.

   ```
   /opt/sun/comms/messaging64/bin/msconfig
   msconfig> set channel:ischedule.official_host_name ischedule-daemon
   msconfig# set channel:ischedule.options.handle-imip 1
   msconfig# set channel:ischedule.options.ischedule-url
   http://<host>:<port>/dav/ischedule/
   msconfig# set
   instance.job_controller.channel_class:ischedule.master_command
   IMTA_BIN:ischedule
   msconfig# set mapping:conversion.rule
   "IN-CHAN=ischedule;OUT-CHAN=*;TAG=*;CONVERT" NO
   msconfig# set mapping:conversion.rule
   "IN-CHAN=*;OUT-CHAN=*;TAG=ISCHEDULE;CONVERT"
   "YES,CHANNEL=ischedule"
   msconfig# set include_conversiontag 2
   msconfig# write
   ```

   Use the host name and alternately the port for the Calendar Server configured for the iSchedule database.

   > **Note**
   > Be sure to add the the trailing forward slash (/) in the `ischedule-url`, otherwise you will receive the error message "HTTP Error 401 Unauthorized."

2. Edit the filters block to specify messages to be processed by iSchedule.

   ```
   msconfig> edit filter
   ```

   The filter block appears in the editor that is the default configured for your login.

3. Add the following lines to create a filter that selects all the messages that have "text/calendar" MIME as an attachment:

```
require ["mime", "environment"];
if allof(environment :is "vnd.sun.destination-channel" ["ims-ms"],
header :mime :anychild :contenttype :is "content-type"
"text/calendar",
NOT header :contains "X-Oracle-CS-iSchedule-Ignore" "Yes") {
addconversiontag "ISCHEDULE";
}
```

iMIP messages generated by Calendar Server contain a "X-Oracle-CS-iSchedule-Ignore: Yes" header to indicate that the event was already added to the user's calendar. So, the Sieve rule should ignore those iMIP messages by not tagging them with an ISCHEDULE conversion tag. Failing to do so results in an iSchedule post of the event that is already present in the user's calendar.

4. Write the configuration and exit the `msconfig` interactive mode.

```
msconfig# write
msconfig> exit
#
```

5. If you are using a compiled configuration, recompile the configuration.

```
/opt/sun/comms/messaging64/bin/imsimta cnbuild
```

6. Restart Messaging Server.

```
/opt/sun/comms/messaging64/bin/stop-msg
/opt/sun/comms/messaging64/bin/start-msg
```

7. Verify the Calendar Server configuration.
   See Verifying the Calendar Server Configuration.

Verifying the Calendar Server Configuration

To ensure that your Calendar Server configuration is setup properly, use the following steps to verify SMTP settings and iMIP email notifications. iMIP email notifications need to also be configured for internal users, that is, users on the same Calendar Server host. If necessary, restart the GlassFish Server container on which Calendar Server is deployed.

1. Check the SMTP configuration for the following settings.

```
cd <cal-svr-base>/sbin
./davadmin config list|grep smtp
notification.dav.smtphost=host2.example.com
notification.dav.smtpuser=user
notification.dav.smtppassword=********
notification.dav.smtpport=25
notification.dav.smtpstarttls=true
notification.dav.smtpusessl=false
notification.dav.smtpdebug=false
notification.dav.smtpauth=false
```

2. Check the email notifications configuration.

```
./davadmin config modify -o notification.dav.smtpstarttls -v false
Enter Admin password:
./davadmin config list|grep imip
notification.dav.enableimipemailnotif=false
./davadmin config modify -o notification.dav.enableimipemailnotif -v
true
Enter Admin password:
```

3. Check the whitelist configuration for the iSchedule port.
   The `service.dav.ischedulewhitelist` configuration parameter prevents denial of service attacks on the iSchedule port. See the topic on enabling the iSchedule channel to handle iMIP messages in *Calendar Server System Administrator's Guide* for more information.
4. If necessary, restart GlassFish Server.
   For example:

```
/opt/SUNWappserver/bin/asadmin stop-domain domain1
/opt/SUNWappserver/bin/asadmin start-domain domain1
```

Modifying iSchedule Channel Options

After you have configured the iSchedule channel, you might need to change iSchedule channel options as described in this section.

Topics in this section:

- To Enable or Disable iMIP Message Processing
- To Modify the iSchedule Service URL

To Enable or Disable iMIP Message Processing

- Use the `msconfig` command to enable or disable iMIP message processing by setting the `handle-imip` option to 1 or 0 respectively.
  For example, the following command disables iMIP message process:

```
/opt/sun/comms/messaging64/bin/msconfig
msconfig> set instance.channel:ischedule.options.handle-imip 0
msconfig# write
msconfig> exit
```

To Modify the iSchedule Service URL

- Use the `msconfig` command to modify the iSchedule URL by editing the `ischedule-url` option.
  For example:

```
/opt/sun/comms/messaging64/bin/msconfig
msconfig> set instance.channel:ischedule.options.ischedule-url
http://<host>:<port>/dav/ischedule/
msconfig# write
msconfig> exit
```

Configuring the iSchedule Channel in Legacy Configuration

This section describes how to configure the iSchedule channel for Messaging Server in legacy configuration (that is, Messaging Server 7 Update 5 and greater but you either did not convert an existing deployment to Unified Configuration, or choose to install a fresh deployment using Unified Configuration.)

1. Create the iSchedule channel.
   a. Add following lines to the *msg-svr-base*`/config/imta.cnf` file:

```
ischedule
ischedule-daemon
```

   b. Add the following lines to the *msg-svr-base*`/config/job_controller.cnf` file:

```
[CHANNEL=ischedule]
master_command=IMTA_BIN:ischedule
```

2. Configure CONVERSION mapping by adding the following lines to the *msg-svr-base* `/config/mappings` file:

```
CONVERSION

 IN-CHAN=ischedule;OUT-CHAN=*;TAG=*;CONVERT NO
 IN-CHAN=*;OUT-CHAN=*;TAG=ISCHEDULE;CONVERT YES,CHANNEL=ischedule
```

3. Enable or disable iMIP message processing.
   To enable or disable iMIP message processing, create a channel options file, *msg-svr-base* `/config/ischedule_option`, and add the following line:

```
handle-imip=1 (to enable)
handle-imip=0 (to disable)
```

4. Configure the iSchedule service URL.
   In the channel options file, specify the iSchedule Service URL as follows:

```
ischedule-url=http://<host>:<port>/dav/ischedule/
```

5. Configure the `include_conversiontag` MTA option if you want to have a {TAG=}} clause included in your conversion mapping probes by adding the following line to the *msg-svr-base* `/config/option.dat` file:

```
INCLUDE_CONVERSIONTAG=2
```

6. Specify messages to be processed by iSchedule.
   Run the following Sieve script to select all the messages that have text/calendar MIME as an attachment. This script should be placed in the location of your system-wide scripts.

```
require ["mime", "environment"];
if allof(environment :is "vnd.sun.destination-channel" ["ims-ms"],
header :mime :anychild :contenttype :is "content-type"
"text/calendar",
NOT header :contains "X-Oracle-CS-iSchedule-Ignore" "Yes") {
addconversiontag "ISCHEDULE";
}
```

7. If you are using a compiled configuration, recompile the configuration.

```
/opt/sun/comms/messaging64/bin/imsimta cnbuild
```

8. Restart Messaging Server.

```
/opt/sun/comms/messaging64/bin/stop-msg
/opt/sun/comms/messaging64/bin/start-msg
```

9. Verify the Calendar Server configuration.
   See Verifying the Calendar Server Configuration

Troubleshooting the iSchedule Configuration

Use the following information to troubleshoot your iSchedule configuration:

- All messages processed through the iSchedule channel have a "Received:" header containing `ischedule-daemon.`*`host`*`.`*`domain`*. This is true whether handling iMIP is enabled or not. If iMIP handing is enabled, the iMIP messages have an extra header, "X-Oracle-CS-iSchedule-Status:," which contains the HTTP status code sent by the iSchedule service in response to the posted iSchedule message.
- The iSchedule channel log files are located in *msg-svr-base*`/log/ischedule_master.log.*`
- Use the *msg-svr-base*`/bin/imsimta qm counters` command to list the number of messages processed by the iSchedule channel.
- One common misconfiguration is to specify a wrong destination channel in the Sieve rule. If you do not see the "X-Oracle-CS-iSchedule-Status:" header in iMIP e-mails, or do not see the iSchedule counters increase when you use the `imsimta qm counters` command, check if the destination channel that you specified in the Sieve rule matches the destination channel that you have configured for that user.

Chapter 54. Performance Tuning DNS Realtime BlockLists (RBL) Lookups

Performance Tuning Realtime BlockLists (RBL) Lookups

The `dns_verify.so` Messaging Server plugin provides a mechanism to block emails based on DNS Realtime Blocklists (RBL) data. RBL Blocklists provided by organisations such as Spamhaus provide an excellent mechanism to reduce the number of emails that are sent from IP addresses of hosts that are known or highly-likely to send spam or bulk unsolicited emails.

This document contains the following sections:

Performance Discussion

The use of DNS RBL lookups to reduce spam email comes at the cost of some additional CPU and network utilisation plus increased time to accept email messages due to DNS resolution delays.

The additional CPU and network utilisation tends to be negated by the overall reduction in email processing due to less spam emails – and therefore less overall emails. The increased time to accept email messages due to DNS resolution delays is a very-real issue that results in a bottleneck in the rate that emails can be accepted.

The most efficient point to see if the IP address of the connecting host is listed in a DNS Realtime Blocklists is at the initial connection state. The PORT_ACCESS mapping table is the first table that is checked, and therefore this is the table most commonly used to perform the dns_verify.so library callout.

In Messaging Server 6.2 and below, the PORT_ACCESS mapping table is only checked by the dispatcher process by default. The dispatcher process uses a single-thread-per-listen-port model e.g. port 25 (SMTP) is one thread, port 587 (SMTP_SUBMIT) is another thread.

As the dispatcher uses a single-thread-per-listen-port, the rate at which an initial email connection can be accepted, compared against the PORT_ACCESS mapping table and then handed off to the multi-threaded tcp_smtp_server process will depend on the time taken for the PORT_ACCESS mapping table comparison to be performed.

Large DNS resolution times in the dns_verify.so callout will therefore cause a bottleneck in the rate connections can be accepted and handed off. The common symptom of this bottleneck is for a system to take a long time to return the initial SMTP banner when the system is either under heavy client connection load or experiencing large DNS resolution times.

Relevant Changes in Messaging Server

Two changes made in Messaging Server 6.3 directly impact the overall performance of dns_verify.so lookups.

Messaging Server 6.3

RFE (Request For Enhancement) #6322877 - "Have SMTP server processes respect the overall result of their PORT_ACCESS probes" was implemented in Messaging Server 6.3. This resulted in the

PORT_ACCESS mapping table being unconditionally checked twice for any given connection. Once in the dispatcher, and a second time in the tcp_smtp_server process.

Two newly documented flags, **$:A** and **$:S**, control whether a PORT_ACCESS rule should only be checked at the dispatcher or tcp_smtp_server level.

As a result of this change, dns_verify.so callouts in the PORT_ACCESS table may be called twice, thus increasing load on DNS resolution infrastructure.

Messaging Server 6.3 (patch 120228-25 and above)

Bug #6590888 - "MS6.3: SMTP server processes not respecting result of PORT_ACCESS probes" was fixed in 120228-25 and above. Prior to this bugfix, it was not possible to have a dns_verify.so callout drop (reject) an email connection if the callout was only performed at the tcp_smtp_server level (i.e. the $:S flag was used).

Hints and Tips

Reduce DNS Lookups

Prevention is better then cure. Careful rearrangement and modification of mapping table rules can assist in reducing the overall number of DNS lookups that are performed and therefore improve the rate that emails can be accepted.

- **Use absolute DNS lookups by adding a "." to the end of the domain**

Using a relative domain lookup e.g. *zen.spamhaus.org* vs. an absolute lookup e.g. *zen.spamhaus.org.* will result in unnecessary lookups. The number of additional lookups will depend on the systems /etc/resolv.conf configuration. A configuration with numerous 'search' domains defined will result in an equivalent number of additional lookups.

(relative domain lookup - a single search domain defined : aus.sun.com)

```
TCP|*|25|*|*
$C$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org,Yours
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
```

```
mailserver.aus.sun.com -> dns.Aus.Sun.COM DNS C
3.100.168.192.zen.spamhaus.org. Internet TXT ?
dns.Aus.Sun.COM -> mailserver.aus.sun.com DNS R  Error: 3(Name Error)
mailserver.aus.sun.com -> dns.Aus.Sun.COM DNS C
3.100.168.192.zen.spamhaus.org. Internet Addr ?
dns.Aus.Sun.COM -> mailserver.aus.sun.com DNS R  Error: 3(Name Error)
mailserver.aus.sun.com -> dns.Aus.Sun.COM DNS C
3.100.168.192.zen.spamhaus.org.aus.sun.com. Internet Addr ?
dns.Aus.Sun.COM -> mailserver.aus.sun.com DNS R  Error: 3(Name Error)
```

(absolute domain lookup - one less lookup compared to relative domain lookup)

```
TCP|*|25|*|*
$C$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Your
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
```

```
mailserver.aus.sun.com -> dns.Aus.Sun.COM DNS C
3.100.168.192.zen.spamhaus.org. Internet TXT ?
dns.Aus.Sun.COM -> mailserver.aus.sun.com DNS R  Error: 3(Name Error)
mailserver.aus.sun.com -> dns.Aus.Sun.COM DNS C
3.100.168.192.zen.spamhaus.org. Internet Addr ?
dns.Aus.Sun.COM -> mailserver.aus.sun.com DNS R  Error: 3(Name Error)
```

- **Restrict the rule to port 25 and non-internal IP addresses (after the INTERNAL_IP)**

To avoid unnecessary lookups for internal systems, place the RBL DNS lookup rule *after* the default INTERNAL_IP PORT_ACCESS rule and restrict the rule to port 25 only as this prevents internal systems from being accidentally blocked and stops email submission (port 587/465) from being checked e.g.

```
PORT_ACCESS

! TCP|server-address|server-port|client-address|client-port
  *|*|*|*|*  $C$|INTERNAL_IP;$3|$Y$E
  TCP|*|25|*|*
$C$:S$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Y
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
  *  $YEXTERNAL
```

- **Use the appropriate mapping table modifier for your version of Messaging Server**

If you have MS6.3 patch **120228-24 or below**
Use $:A to halve the number of lookups e.g.

```
TCP|*|25|*|*
$C$:A$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Y
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
```

If you have MS6.3 patch **120228-25 and above**
Use $:S to move lookups to the multi-threaded smtp-server process

```
TCP|*|25|*|*
$C$:S$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Y
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
```

(**Not** using $:S or $:A modifier - twice the number of lookups)

```
02:30:15.629216 IP mailserver.Aus.Sun.COM.41249 >
dns.Aus.Sun.COM.domain: 27201+ TXT? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.629222 IP mailserver.Aus.Sun.COM.41249 >
dns.Aus.Sun.COM.domain: 27201+ TXT? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.631251 IP dns.Aus.Sun.COM.domain >
mailserver.Aus.Sun.COM.41249: 27201 NXDomain 0/1/0 (110)
02:30:15.631474 IP mailserver.Aus.Sun.COM.41250 >
dns.Aus.Sun.COM.domain: 27202+ A? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.631480 IP mailserver.Aus.Sun.COM.41250 >
dns.Aus.Sun.COM.domain: 27202+ A? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.632386 IP dns.Aus.Sun.COM.domain >
mailserver.Aus.Sun.COM.41250: 27202 NXDomain 0/1/0 (110)
02:30:15.633410 IP mailserver.Aus.Sun.COM.41251 >
dns.Aus.Sun.COM.domain: 28805+ TXT? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.633418 IP mailserver.Aus.Sun.COM.41251 >
dns.Aus.Sun.COM.domain: 28805+ TXT? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.634324 IP break.Aus.Sun.COM.domain >
mailserver.Aus.Sun.COM.41251: 28805 NXDomain 0/1/0 (110)
02:30:15.634526 IP mailserver.Aus.Sun.COM.41252 >
dns.Aus.Sun.COM.domain: 28806+ A? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.634531 IP mailserver.Aus.Sun.COM.41252 >
dns.Aus.Sun.COM.domain: 28806+ A? 3.100.168.192.zen.spamhaus.org. (46)
02:30:15.635325 IP break.Aus.Sun.COM.domain >
mailserver.Aus.Sun.COM.41252: 28806 NXDomain 0/1/0 (110)
```

(Using $:S or $:A modifier)

```
02:32:07.923587 IP mailserver.Aus.Sun.COM.41253 >
dns.Aus.Sun.COM.domain: 63100+ TXT? 3.100.168.192.zen.spamhaus.org. (46)
02:32:07.923599 IP mailserver.Aus.Sun.COM.41253 >
dns.Aus.Sun.COM.domain: 63100+ TXT? 3.100.168.192.zen.spamhaus.org. (46)
02:32:07.924979 IP dns.Aus.Sun.COM.domain >
mailserver.Aus.Sun.COM.41253: 63100 NXDomain 0/1/0 (110)
02:32:07.927616 IP mailserver.Aus.Sun.COM.41254 >
dns.Aus.Sun.COM.domain: 63101+ A? 3.100.168.192.zen.spamhaus.org. (46)
02:32:07.927627 IP mailserver.Aus.Sun.COM.41254 >
dns.Aus.Sun.COM.domain: 63101+ A? 3.100.168.192.zen.spamhaus.org. (46)
02:32:07.928609 IP dns.Aus.Sun.COM.domain >
mailserver.Aus.Sun.COM.41254: 63101 NXDomain 0/1/0 (110)
```

- **Place rate-limiting mechanisms (metermaid, conn_throttle, and so on) before DNS RBL lookups**

If you use one of the email rate-limiting mechanisms, such as metermaid or `conn_throttle.so`, placing these PORT_ACCESS rate-limiting lookups *prior* to the `dns_verify.so` lookup will help reduce the impact of a Denial of Service on Messaging Server, for example:

```
PORT_ACCESS

! TCP|server-address|server-port|client-address|client-port
  *|*|*|*|*  $C$|INTERNAL_IP;$3|$Y$E
  *|*|*|*|*
$C$:A$[IMTA_LIB:check_metermaid.so,throttle,ext_throttle,$3]$N421$
Connection$ declined$ at$ this$ time$E
  TCP|*|25|*|*
$C$:S$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Y
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
  *  $YEXTERNAL
```

- **Use most successful lookup first (multiple lookups)**

By placing the RBL lookups in most-successful to least-successful order, the overall number DNS lookups will be reduced as Messaging Server will terminate the PORT_ACCESS mapping table processing after the first RBL lookup returns a DNS TXT or A record.

Adding the "$T" PORT_ACCESS mapping table flag to the dns_verify.so callout will provide additional logging information to help determine which RBL is the most successful e.g.

```
TCP|*|25|*|*
$C$:S$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Y
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
```

Adding "LOG_CONNECTION=7" to the Messaging Server MTA *option.dat* configuration file will result in an additional "T" record in the mail.log file when a connection is dropped due to the connecting host being listed in a DNS RBL e.g.

```
12-Mar-2008 10:06:52.09 78f.4.686597 **          +      T
TCP|1.2.3.4|25|5.6.7.8|39802 571
http://www.spamhaus.org/query/bl?ip=5.6.7.8
```

In the above case the SpamHaus lookup returned a TXT record "http://www.spamhaus.org/query/bl?ip=5.6.7.8" which was returned to the connecting client.

By using this log information, and re-ordering the DNS RBL lookups to provide the best first-lookup match rate, your DNS lookups will be reduced therefore improving overall performance.

The following site offers a comparison of the performance of various common RBL Blacklist providers which should provide a good guide on where to start ordering the lookups:

http://stats.dnsbl.com/

- **Don't use too many lookups**

If your site uses multiple DNS RBL lookups to increase the chances of blocking IP addresses that are known to send spam, reordering those rules as per the previous Hint/Tip may show that the latter lookups block negligible additional hosts and can therefore be removed.

- **Don't use DNS_VERIFY_DOMAIN dispatcher configuration**

The DNS_VERIFY_DOMAIN dispatcher option doesn't provide sufficient granularity and therefore dns_verify.so PORT_ACCESS lookups should be used instead as discussed throughout this guide.

- **Avoid lookups for known 'Friendly' IP ranges**

IP addresses for an organisation can usually be split into three distinct categories:

=> Internal IP addresses of trusted email upload systems (e.g. other Messaging Server MTA relays, mailstores). These are usually defined in the INTERNAL_IP mapping table and which you never want to be blocked if the IP address of a host happens to be listed in Realtime Blacklist.

=> 'Friendly' IP addresses of trusted hosts which your organisation has direct control over (e.g. user's PC), and can therefore take action to quarantine if the systems are found to be a source of spam email. These systems are unlikely to ever be listed on a DNS RBL blacklist and if they are you don't want them to be blocked. They are not trusted enough to consider 'Internal'.

=> External IP addresses which cannot be trusted and whose IP addresses definitely need to be verified against Realtime Blacklists.

To define a range of 'Friendly' IP addresses, add a new mapping table called FRIENDLY_IP, this table will have the same format as the INTERNAL_IP mapping table e.g.

```
FRIENDLY_IP

  $(192.168.100.0/24)  $Y
  *  $N
```

Add a new 'FRIENDLY_IP' check to the PORT_ACCESS mapping table. This check should be above any dns_verify.so lookups, but below any rate-limiting checks (to protect Messaging Server from Denial of Service attacks) e.g.

```
PORT_ACCESS

! TCP|server-address|server-port|client-address|client-port
  *|*|*|*|*  $C$|INTERNAL_IP;$3|$Y$E
  *|*|*|*|*
$C$:A$[IMTA_LIB:check_metermaid.so,throttle,ext_throttle,$3]$N421$
Connection$ declined$ at$ this$ time$E
  *|*|*|*|*  $C$|FRIENDLY_IP;$3|$YEXTERNAL$E
  TCP|*|25|*|*
$C$:S$[IMTA_LIB:dns_verify.so,dns_verify_domain_port,$1,zen.spamhaus.org.,Y
host$ ($1)$ found$ on$ spamhaus.org$ RBLblock$ list]$T$E
  *  $YEXTERNAL
```

- **Have customers use 'submit' port or SSL port for sending emails**

The dns_verify.so lookups used in this guide are restricted to port 25 server connections only.

If a customer uploading emails (e.g. using Mozilla Thunderbird) to your Messaging Server uses a submit port (e.g. port 587) they will avoid the DNS RBL lookup – although they will still be required to authenticate so this mechanism does not provide a means of spammers to easily bypass the RBL checks.

Using the 'submit' port reduces the number of RBL checks that need to be performed and also stops your email customers from being accidentally blocked.

Improve Performance of DNS Lookups

If you need to perform a DNS RBL lookup, they should be as fast-as-possible to reduce the impact on overall email delivery and processing performance.

- **Use a local-caching name-server process**

A local-caching name-server will keep a local cache of DNS lookups; thus reducing network overhead (and delay) and reducing the impact of any network infrastructure/DNS infrastructure problems.

The following guides provide information on how to install and configure Bind 9 on the messaging server to operate as a caching name-server.

http://www.brandonhutchinson.com/Caching-only_BIND_nameserver.html
http://www.learning-solaris.com/index.php/configuring-a-dns-server/
http://www.logiqwest.com/dataCenter/Demos/RunBooks/DNS/DNSsetup.html

- **Use local copy of Realtime Blacklist DNS tables**

Organisations such as SpamHaus provide the option to keep a local copy of the RBL DNS tables. This can then be used to provide a local copy of the RBL Blacklist data which is much faster and potentially more reliable then relying on an external DNS servers.

spamhaus.org data feed
njabl.org data feed
sorbs.net data feed
cbl.abuseat.org data feed (note: CBL data already included in zen.spamhaus.org)
dsbl data feed
spamcop.net data feed

- **Use fast and reliable Realtime Blacklist DNS providers only**

Smaller DNS Realtime Blacklist providers may not have sufficient or local DNS mirrors to provide quick lookup times or they may be prone to periods of outages when heavily loaded.

Prior to using **any** RBL, make sure you search the Internet using your preferred web-search engine for any existing reviews, problems etc.

The consequences of an incorrect choice can be severe.

For example the ordb.org RBL list shutdown in 2006. System administrators that didn't notice that the ordb.org list was no longer blocking emails received a rude shock on the 25th March 2008 when lookups using the ORDB list now returned a successful value for *all* lookups – therefore causing all emails to be blocked as a result.

Chapter 55. Protecting Against Spammers who Compromise Messaging Server User Accounts

Best Practices for Protecting Against Spammers who Compromise Oracle Communications Messaging Server User Accounts

Topics:

Overview of Email Spammers and Compromised User Accounts

Spammers are now using sophisticated phishing attacks to target individual organizations and collect valid login details from ill-informed and overly-trusting account owners. Phishers then use these compromised account details to send spam emails by authenticating to the Messaging Server MTA and Webmail processes, thus bypassing this security restriction.

As the spam emails are delivered external recipients, Realtime Black-list's (RBLs) are listing these sending organizations. This in turn is causing legitimate non-spam emails to be rejected by organizations that use these same RBL's.

This document provides best-practice information on how to protect your organization against phishers and compromised user accounts. It provides proactive and reactive methods to reduce the impact of compromised accounts.

Preventing Outbound Spam: Proactive Methods

Reduce the chances that a targeted phishing attack succeeds by implementing preventative measures such as:

- Educating your account holders. This is the best method to proactively avoid problems. For example, send regular reminders that your organization will **never** ask for account details by using email, and that users need to immediately report such emails. Set up an appropriate role account for this task.
- Implementing anti-spam and anti-virus applications that check for phishing style email. For more information, see Planning a Messaging Server Anti-Spam and Anti-Virus Strategy.
- Blocking known phishing addresses or common role accounts from sending emails from outside

the organization, for example, `helpdesk@domain.com`, `security@domain.com`, and so on. See http://code.google.com/p/anti-phishing-email-reply/ for more information.
- Using good password policies. Stop easy-to-guess passwords (this includes administration accounts and role accounts, that is, `uid=admin`, `calmaster`, and so on) to protect against dictionary attacks. Use password expiry to force users to change passwords on a regular basis.
- Authenticated emails with different From: addresses (especially if not for the organization) increase the chances that your email accounts are used for sending out spam, or indeed used for additional phishing attacks against other organizations.

Preventing Outbound Spam: Reactive Measures

Despite the best preventative measures, spammers can still acquire valid account details. By putting in place mechanisms to limit the number of email messages that users can send, you reduce the impact of compromised accounts. You should use these limiting techniques on both outgoing and incoming email.

Blocking Submissions of Local Senders Who Might Be Spammers

> The features described in this section were introduced in **Messaging Server 7 Update 2**

When a compromised user account is used to send emails to a large number of external email addresses, it is highly probable that a number of these email addresses will be invalid or trigger spam filtering mechanisms at the recipient server end and be rejected. Through the use of the `LOG_ACTION` mapping table and metermaid, it is possible to restrict email upload based on these rejections. Further details are available in the Blocking submissions of local senders who may be spammers section of the Triggering Effects From Transaction Logging: The LOG_ACTION Mapping Table documentation.

Rate Limiting All Outgoing Email

Rate limit outgoing email as shown in this example. Use different levels of restrictions depending on the "trust" of the IP address of the client sending the email, for example:

```
=> most emails for internal auth-send
=> less emails for internal non-auth-send
=> less emails again for external-auth-send
=> less emails again for mshttpd source (Webmail emails) for practical reasons a human cannot send a lot of emails via Webmail in a short period of time.
```

Rate Limiting Only Outgoing Spam

Implement scanners/spam filtering on outgoing email. One idea is to use a spam filter, such as SpamAssassin, to flag messages as 'spammy' and call a Sieve action that calls a mapping rule, which calls MeterMaid to monitor the count of these emails (on env-from address). If the number of emails exceeds some threshold then perform an action on the email, for example hold, capture a copy, discard, bounce, and so forth.

For more information on this technique, see Can Sieve Scripts Use Functions Defined Via Mapping Tables?

An example:

Configure your anti-spam scanner to add an X-header to all outbound messages that indicates whether the message is spam. E.g.

```
X-Spam-Score-Internal: ****
```

Add the following to a channel that processes your outbound mail. This will cause a sieve filter to be executed for all messages dequeued from that channel. You can also use a destinationfilter, depending on your environment.

```
 sourcefilter file:IMTA_TABLE:authspam.filter
```

Create a sieve filter called "authspam.filter" in your config directory. It checks to see if the message is rated as spam (from the X-header) and it extracts the env-from and env-to from the message. It makes a call to a mappings table with the env-from and the env-to as arguments. It then rejects the message back to the env-from if it gets a positive response from the mappings.
The next step after identifying a compromised account is to prevent further misuse of the account by spammers and address any negative consequences e.g. being listed on blacklist. The following techniques will provide a starting point:

```
require ["variables","reject","envelope"];

# only limit messages rated as spam
if header :contains "X-Spam-Score-Internal" "*****" {

    # pull out the envelope from address
    if envelope :all :matches "from" "*" {
        set "FROM_ADDR" "${1}";

        # pull out the envelope to address
        if envelope :all :matches "to" "*" {
            set "TO_ADDR" "${1}";

            # perform FILTER_limitauthspam mapping callout
            if limitauthspam "${FROM_ADDR}|${TO_ADDR}" {
                set "RESULT" "${0}";

                # reject the message
                reject "Your account has been sending a lot of messages that
appear to be spam. ";
            }
        }
    }
}
```

Put this in the mappings. The sieve script makes a call to this mapping to query metermaid. The mapping includes exemptions if the env-to matches recipients that you want to be able to recieve spam messages from your users.

```
FILTER_limitauthspam

*|is-spam@*                    $N$E
*|not-spam@*                   $N$E
*|abuse@*                      $N$E
*|postmaster@*                 $N$E
*|*      $[IMTA_LIB:check_metermaid.so,throttle,limitauthspam,$0]$0$Y
```

Set these options in configutil to enable the metermaid database. This will cause metermaid to allow 50 outbound spam messages per hour for each env-from address.

```
metermaid.table.limitauthspam.data_type = string
metermaid.table.limitauthspam.quota = 50
metermaid.table.limitauthspam.quota_time = 3600
metermaid.table.limitauthspam.options = nocase
metermaid.table.limitauthspam.max_entries = 1000
```

Note: This won't work if the messages aren't rated as spam. 419 scams are notorious for slipping through spam filters.

It's possible for the spammer to forge their env-from address. If this occurs, the sieve will need to be updated to accommodate. Or, do not allow outgoing email with a different From: address.

Can Sieve Scripts Use Functions Defined Via Mapping Tables?

Yes, Sieve scripts can use functions defined through mapping tables. In particular, a Sieve script can use a function that is implemented as a mapping table that does a `dns_verify` routine call to query spamhaus.

Although this mapping-table-as-Sieve-function functionality has existed for quite awhile, the syntax doesn't settle down until Messaging Server 6.3. For Messaging Server 6.2, you can do this with a bit of an incompatibility in the variables usage. (This document does not address releases prior to Messaging Server 6.2.)

- Starting with Messaging Server 6.3, to have a Sieve function that is a mapping table, create a mapping table named `FILTER_`*function-name*, for example:

  ```
  FILTER_foo

    a     b$Y
  ```

  Starting with Messaging Server 6.3, Sieve script can then accomplish something like the following:

  ```
  require "variables";
  if foo "a" {set "foo-result" "{$0}";
              set "foo-flags" "{$1}";
              if string :is "${foo-result}" "b" { discard; }
             }
  ```

  This Sieve script sets `foo-result` to **b** and `foo-flags` to **Y**, and discards the incoming message.

- In Messaging Server 6.2, to have a Sieve function that is a mapping table:

```
require "variables";
if foo "a" {set "foo-result" "{$1}";
            set "foo-flags" "{$2}";
            if string :is "${foo-result}" "b" { discard; }
           }
```

  Note the different variable numbering.

In general, the idea is to have a mapping table (say `FILTER_spamhaus`) that uses a `dns_verify` callout to query spamhaus. Have your Sieve script check for relevant header lines that might have a source IP, then call the spamhaus function (which is the `FILTER_spamhaus` mapping) passing in to the function/mapping the source IP your script found (and stored in one variable) and getting back a spamhaus result in another variable; check that returned variable and based upon it do something useful, whatever you consider useful: discard the message, reject it, .HELD it with the "hold" action, do a "spamadjust", and so on.

Reject/Discard All Outbound Spam

If your tolerance for outbound spam is high, and you don't care about the occasional message being blocked by your spam filter, rejecting or discarding all outbound spam message back to the sender is an effective way to deal with the event of a compromised account.

You may want to disable "IP reputation checks" in your spam scanner for when it processes your outbound mail since many consumer IPs will be on black lists.

If you are rejecting the messages back to the sender, be careful that you are only rejecting mail to authenticated senders. If you want to prevent outbound mail that you are forwarding, then you should not reject the mail since it will backscatter out to the internet and get your servers blacklisted. Consider discarding or quarantining this mail instead.

Recovering From Phishing Attacks That Have Compromised User Accounts

The next step after identifying a compromised account is to prevent further misuse of the account by spammers and address any negative consequences e.g. being listed on a real-time blacklist. The following techniques will provide a starting point:

- Prevent further logins of the comprised user account:
  - Mark account as inactive (`mailUserStatus: inactive`).
  - Change the password of the account.
  - Advise the local IT support helpdesk that access to the account has been blocked so that should the owner contact the IT help desk they can work with the customer to use improved password policies, and so on, in the future.
- Kill any existing logins by using the `./imsconnutil -k -u uid` command (starting with Messaging Server 6.3).
- Block the IP address used to send the email at your network firewall.
- Kill any existing webmail sessions to prevent re-use.
  - Increase logging to Information. This is required to capture the session ID information:

```
./configutil -o logfile.http.loglevel -v Information
```

  - Disable http IP security. With IP security enabled, only the IP address that initially logged into the webmail process will be able to logout.

```
./configutil -o service.http.ipsecurity -v no
```

- Restart mshttpd processes.

```
./stop-msg http;./start-msg http
```

- If you find an account is compromised, locate the login string with the SID (session ID), for example:

```
[05/May/2009:12:23:21 +1000] server httpd[7257]: Account
Information: login [129.158.87.204:51539] user001
plaintext sid=YvgZdFHgwx0
```

- Change/reset the password for the compromised account.
- Use `wget` to log out the session:

```
wget -o /dev/null "https://<server name>/cmd.msc?sid=<session
ID>&cmd=logout"
for example:
wget -o /dev/null
"https://server.aus.sun.com/cmd.msc?sid=YvgZdFHgwx0&cmd=logout"
```

- Find and remove any existing spam email sent via the compromised account in the tcp_local MTA queue.
- Find out if you have been black-listed: spamcop, Realtime Blackhole List Lookup.
  - To remove yourself from a blacklist depends on the list. For example, see spamhaus.org and mail-abuse removal request.
- Vary the IP address of your outgoing smtp client for the tcp_local channel.
  - Bind outgoing email to a IP address by using the interfaceaddress SMTP channel option.
  - If an IP address gets blacklisted, shift to another IP address (be careful if you are using SPF).
- Enable Directory Server audit log: monitor for changes, such as signature files and reply-to address, by using a script and crontab to classify likely compromised accounts; remove modifications.

Greylisting Webmail

The following proof-of-concept instructions describe how to enable greylisting of emails that are sent through the webmail process (Messaging Express, Communications Express, or Convergence). You use the third-party gross daemon and plug-in to provide greylisting functionality. One advantage of the gross daemon is that you can configure greylisting only if the sender's IP address is also on a blacklist.

Requirements for Greylisting Webmail

The software requirements for configuring greylisting on webmail are:

- Gross Daemon: You can run the daemon on the same server as Messaging Server or on another server.
- At least Messaging Server 7.0 MTA: Messaging Server 7.0 inclues the ereject Sieve functionality that you use to cause emails to be rejected.
- At least Communications Express 6.3p4: This version contains a fix for CR 6648111 (UWC should pass client IP address to `mshttpd` for use in mail headers).
- At least Messaging Server 6.3: This version contains a fix for RFE 6424798 (Have MEM recognize

"Proxy-ip: header for use in passing client IP back to `mshttpd`).

Installing and Configuring Greylisting for Webmail

1. Download, compile, configure and start the gross daemon.
   See Quick Start Guide and http://code.google.com/p/gross/.
2. Copy the `grosscheck.so` library file, compiled as part of the compilations of the gross daemon, to the MTA server's *msg_base*`/lib` directoy.
3. Compile and install the `c-ares` library on the MTA server.

> **Note**
> If the platform that is running the gross daemon is different from the MTA server platform, recompile the gross daemon to get a compatible`grosscheck.so` library.

4. Configure the Messaging Server 7.0 MTA by creating a new file in the *msg_base*`/config` directory called `greylist.sieve` containing the following code:

```
require ["ereject","variables"];
# Need to extract IP address from Received Header
# Require UWC6.3p4 and above to add the Forward-For: header
if (header :matches "Received" "*(Forwarded-For: *)*") {
   set "IP_ADDR" "${2}";

   # Need to extract header from address
   if (header :matches "From" "*<*>*") {
       set "FROM_ADDR" "${2}";

       # Perform FILTER_GREYLIST mapping callout
       set "RESULT" greylist("${IP_ADDR}|${FROM_ADDR}|uwc");

       # Block if greylist check returns a value -- indicating that
they are a 'bad' sender
       if (not string :is "${RESULT}" "")
         { # *NOTE* erejec is used instead of erejec(t) to workaround
bug
#6704720
           erejec "Delivery failed. Please wait 10 seconds and try
sending again...";
       }
   }
}
```

1. Add the following to the *msg-svr-base*`/config/mappings` file:

```
FILTER_GREYLIST

 ! use gross to check all triplets (client_ip,sender,recipient)
 *|*|*
$C$[IMTA_LIB:grosscheck.so,grosscheck,129.158.87.192,,5525,$0,$1,$2,]$Y$E
* $Y
```

2. Add the following to the source channel in the *msg-svr-base*`/config/imta.cnf` file that accepts email from the `mshttpd` process, that is, `tcp_intranet`, `tcp_uwc`:

```
sourcefilter file:IMTA_TABLE:greylist.sieve
```

3. Recompile and restart the MTA.
   `./imsimta cnbuild;./imsimta restart`

Troubleshooting Your Greylisting Deployment

1. Configure the `gross.conf` file to use a blacklist that returns a result for *all* IP addresses, for example:

   ```
   dnsbl = relays.ordb.org
   ```

2. Run the `grossd` process in the foreground, for example:
   `./gross -d`
3. Attempt to send a test email.
   You should see message similar to the follwing in the gross output:

   ```
   Fri Jul 18 16:34:53 2008 #9: a=greylist d=2 w=1 c=129.158.87.66
   s=uwc r=user@example.com m=relays.ordb.org+1
   ```

   Webmail users should receive an error message in their email client such as:

   ```
   SMTP Error 5.7.1 Delivery Failed. Please wait 10 seconds and try
   sending again
   ```

   If users receive such an error, instruct them to Click OK, wait 10 seconds, then click the Send button again. The following message should then appear in the gross output:

   ```
   Fri Jul 18 16:42:48 2008 #a: a=match d=0 w=0 c=129.158.87.66 s=uwc
   r=user@example.com
   ```

   The email should also be accepted.

Chapter 56. Rate-limiting Email

Rate-limiting Email

This information describes how to rate-limit, or throttle, email sent from the Communications Express and Messenger Express webmail clients.

Topics:

Overview of Rate-limiting

Rate-limiting, or throttling, of email is usually based on an email client's IP address. This enables you to limit the client's exposure to Denial of Service attacks from external email servers. When it comes to limiting emails from Messenger Express and Communications Express clients, all email is "seen" to come from a single IP address, even though any number of different users could have sent the email. Restricting based on IP address is therefore inappropriate as it offers insufficient granularity.

The introduction of MeterMaid in Messaging Server 6.3 provided the mechanism to restrict based on the original user who composed the email in the webmail interface.

The steps that follow were verified on Messaging Server 6.3 at patch level 120229-23 (Oracle Solaris SPARC x86).

> **Note**
> MeterMaid is currently only able to collect and act on a per-MTA basis. If you use a load-balanced pool of MTA servers for accepting webmail email, you are not able to restrict as precisely, unless MeterMaid is configured to use a single MeterMaid server (which introduces a single-point-of-failure).

Isolating Traffic From Messenger Express and Communications Express

Isolating Messenger Express and Communications Express sourced email provides the following advantages:

- It is easier to monitor and correlate Messenger Express and Communications Express sourced email traffic. Emails submitted by using Messenger Express and Communications Express have a source channel of `tcp_webmail` in the `mail.log` file.
- Email submission is not affected by standard traffic load. Delays that result from a massive number of email being sent from other servers to port 25 no longer affect email submissions from Messenger Express and Communications Express due to sharing the same single-threaded

dispatcher port.

- Change control. You can revert to the previous configuration by changing the webmail port. This limits the impact that the change has on your existing email delivery environment.
- Custom restrictions. If you wanted to further restrict the number of recipients per email sent by Messenger Express and Communications Express without impacting other clients such as Mozilla Thunderbird (which uses a different submission port), you could apply the restriction to just the new `tcp_webmail` source channel.

To isolate the traffic you need to create a new dispatcher listener on port 3025 and have connections to that port configured as the new source channel `tcp_webmail`. For example:

- Add the following to the `dispatcher.cnf` file.

```
!
! UWC upload SMTP server
!
[SERVICE=SMTP_WEB]
PORT=3025
IMAGE=IMTA_BIN:tcp_smtp_server
LOGFILE=IMTA_LOG:tcp_webmail_server.log
PARAMETER=CHANNEL=tcp_webmail
STACKSIZE=2048000
! Uncomment the following line and set INTERFACE_ADDRESS to an
appropriate
! host IP (dotted quad) if the dispatcher needs to listen on a
specific
! interface (e.g. in a HA environment).
!INTERFACE_ADDRESS=
```

- Create a new MTA channel by adding the following to the `imta.cnf` file.

```
!
! tcp_webmail
tcp_webmail smtp missingrecipientpolicy 4
tcp_webmail-daemon
```

Creating a MeterMaid Configuration

1. Enable MeterMaid as described in Throttling Incoming Connections by Using MeterMaid.
   For example:

```
./configutil -o local.metermaid.enable -v yes
./configutil -o metermaid.config.secret -v password
./configutil -o metermaid.config.serverhost -v localhost
./start-msg metermaid
```

2. Add a throttle table for Metermaid.
   For example:

```
./configutil -o metermaid.table.webmail_msg_throttle.data_type -v
string
./configutil -o metermaid.table.webmail_msg_throttle.quota -v 5
./configutil -o metermaid.table.webmail_msg_throttle.options -v
nocase
./configutil -o metermaid.table.webmail_msg_throttle.quota_time -v
1800
```

The preceding table restricts email delivery to a rate of five emails every 1800 seconds.

Creating Mappings Rules

1. To block the number of emails by a given user, add the following to the mappings table.

```
FROM_ACCESS

! restrict number of emails sent per-user in uwc
!
port-access-probe-info|app-info|submit-type|src-channel|from-address|a
*|SMTP*|*|tcp_webmail|*|*
$C$[IMTA_LIB:check_metermaid.so,throttle,webmail_msg_throttle,$3]\
$NExcessive$ email$ sent$ -$ Please$ try$ again$ later$E
```

2. Recompile the MTA configuration and restart

```
./imsimta cnbuild; ./imsimta restart
```

Testing and Debugging the Configuration

1. Enable MeterMaid debug logging.

```
./configutil -o logfile.metermaid.loglevel -v Debug
./stop-msg metermaid; ./start-msg metermaid
```

2. Send a test email and ensure that it increments.
   Send test emails by using an email client such as Mozilla Thunderbird to the MTA on port 3025.

```
[18/Oct/2007:09:27:07 +1000] meg [27419]: General Information: Log
created (1192663627)
[18/Oct/2007:09:27:07 +1000] meg [27419]: General Notice: Creating
table
"webmail_msg_throttle" - type=throttle, data_type=string,
storage=hash, max_entries=1000,
quota=5, quota_time=1800, options=nocase
[18/Oct/2007:09:27:07 +1000] meg [27419]: General Notice: MeterMaid
build date: Aug  3
2007 17:13:42
[18/Oct/2007:09:27:07 +1000] meg [27419]: General Information:
Binding to 0.0.0.0 on
port 63837
[18/Oct/2007:09:27:07 +1000] meg [27419]: General Notice: Ready!
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Information: (1)
Connection accepted
from 127.0.0.1
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Debug: (1)
Received: "HELLO password"
from client
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Information: (1)
Connection
authenticated
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Debug: (1)
Sending: "+ Welcome!"
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Debug: (1)
Received: "CONNECT
webmail_msg_throttle some.user@mydomain.com" from client
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Debug: (1)
command=CONNECT,
table=webmail_msg_throttle, argument=blah
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Information: (1)
Current status for
"some.user@mydomain.com" in webmail_msg_throttle: 1 / 5
[18/Oct/2007:09:27:26 +1000] meg [27419]: General Debug: (1)
Sending: "+ OK"
```

Enable Rate-limiting at the Communications Express Level

1. Change the port that Communications Express and Messenger Express interfaces use to upload email.

```
./configutil -o service.http.smtpport -v 3025
./stop-msg http; ./start-msg http
```

2. To backout the change, restore `service.http.smtpport` to the previous setting and restart the http daemon.

```
./configutil -o service.http.smtpport -v 25
./stop-msg http; ./start-msg http
```

Restricting Access to New Source Channel

Once you are satisfied that the rate-limiting is working, restrict access to port 3025 to only those hosts running the webmail server. This can be achieved by adding a rule to the PORT_ACCESS mappings table, that is, PORT_ACCESS.

```
PORT_ACCESS

! Restrict access to port 3025 to just webmail servers
! TCP|server-address|server-port|client-address|client-port
  TCP|*|3025|127.0.0.1|*                $Y
  TCP|*|3025|10.2.3.5|*                 $Y
  TCP|*|3025|*|*                        $N550$ Access$ denied$ to$
Webmail$ submission$ port
```

This rule rejects any connections to port 3025 that do not come from localhost (127.0.0.1) and another webmail server with an IP address of 10.2.3.5.

Advanced Techniques

Scaling MeterMaid

By default, each MeterMaid table holds only 1,000 entries. If your environment is likely to have more then 1,000 senders in a given time period (quota_time), then scale the max_entries per-table parameter as appropriate. See *Messaging Server Administration Reference* for more information on MeterMaid parameters for more information.

Restricting Based on Recipients

1. Add a mapping rule to the ORIG_SEND_ACCESS mappings table, that is, ORIG_SEND_ACCESS.

   ```
   ORIG_SEND_ACCESS

   ! restrict number of total recipients sent to by a user
   ! src-channel|from-address|dst-channel|to-address
     tcp_webmail|*|*|*
   $C$[IMTA_LIB:check_metermaid.so,throttle,webmail_rcpt_throttle,$0]\
   $NExcessive$ email$ sent$ -$ Please$ try$ again$ later$E
   ```

2. Add a new throttle table as well.

   ```
   ./configutil -o metermaid.table.webmail_rcpt_throttle.data_type -v
   string
   ./configutil -o metermaid.table.webmail_rcpt_throttle.quota -v 1000
   ./configutil -o metermaid.table.webmail_rcpt_throttle.options -v
   nocase
   ./configutil -o metermaid.table.webmail_rcpt_throttle.quota_time -v
   600
   ```

   This table restricts email delivery to a rate of 1000 recipients total every 600 seconds.

Chapter 57. Setting Up and Managing Messaging Server Security

Setting Up and Managing Messaging Server Security

This information provides an overview about security for the Messaging Server product. It also provides links to security topics that provide more in-depth information for configuring and administering Messaging securely.

Topics:

Overview of Messaging Server

For an overview of the product, see the topic on introduction to Messaging Server software in *Unified Communications Suite Deployment Planning Guide*. For information on general security principles, such as security methods, common security threats, and analyzing your security needs, see the topic on designing for security in *Unified Communications Suite Deployment Planning Guide*. For an overview of operating system security, see Oracle Solaris Security for System Administrators.

To see Messaging Server's high-level architecture, see the topic on developing a Messaging Server architecture in *Unified Communications Suite Deployment Planning Guide*. In addition, the following illustration describes the architecture of the Message Transfer Agent (MTA), Messaging Server's message router: MTA Architecture.

Secure Installation and Configuration

This section describes the installation considerations and recommendations for securing your Messaging Server deployment:

Installation Overview

Understanding Your Environment

To better understand your security needs, ask yourself the following questions:

1. Which resources am I protecting?
   In a Messaging Server production environment, consider which of the following resources you want to protect and what level of security you must provide:
   - Messaging Server front end servers
   - Messaging Server back end servers
   - Dependent resources

2. From whom am I protecting the resources?
   In general, resources must be protected from everyone on the Internet. But should the Messaging Server deployment be protected from employees on the intranet in your enterprise? Should your employees have access to all resources within the environment? Should the system administrators have access to all resources? Should the system administrators be able to access all data? You might consider giving access to highly confidential data or strategic resources to only a few well trusted system administrators. On the other hand, perhaps it would be best to allow no system administrators access to the data or resources.
3. What will happen if the protections on strategic resources fail?
   In some cases, a fault in your security scheme is easily detected and considered nothing more than an inconvenience. In other cases, a fault might cause great damage to companies or individual clients that use Messaging Server. Understanding the security ramifications of each resource help you protect it properly.

Deployment Topologies

You can deploy Messaging Server on a single host or on multiple hosts, splitting up the components into multiple front-end Messaging Server hosts and multiple back-end hosts.

The general architectural recommendation is to use the well-known and generally accepted Internet-Firewall-DMZ-Firewall-Intranet architecture. For more information on addressing network infrastructure concerns, see the topic on determining your Communications Suite network infrastructure needs in *Unified Communications Deployment Planning Guide*.

The following guidelines provide specific recommendations for Messaging Server:

- Securing Your Firewall/DMZ architecture
- Using a Firewall to Allow Connections
- Planning Secure High Availability and Load Balancing for your Deployment
- Minimizing Operating System Security Risks

Securing Your Firewall/DMZ architecture

Secure your Messaging Server infrastructure by determining your firewall/DMZ architecture. See *Unified Communications Suite Deployment Planning Guide* for more information.

> **Note**
> Your firewall/DMZ architecture solution might depend on your anti-spam solution and client capabilities. How you handle firewall and DMZ architecture depends on requirements for a geographically dispersed deployment and whether your deployment is targeted at individual end users or enterprises.

Using a Firewall to Allow Connections

Because the Webmail server (`mshttpd`) supports both unencrypted and encrypted (SSL) communication with mail clients, you might use a firewall between your Messaging Store and your mail clients for added security.

Some guidelines to consider when using a firewall:

- If using firewall only allow Convergence server to connect to `mshttpd` (8990, 8991).
- If using firewall (preferably whitelist-based for Messaging Servers), verify internal service protocols are blocked (watcher 49994, job_controller 27442, ENS 7997, third-party authentication server, msadmind 7633, LMTP, Metermaid, and JMS).

Planning Secure High Availability and Load Balancing for your Deployment

See Configuring Messaging Server for High Availability and the topic on designing for service availability in *Unified Communications Suite Deployment Planning Guide* for how to set up a secure high availability and load balancing Messaging Server deployment.

Minimizing Operating System Security Risks

See the topic on operating system security in *Unified Communications Suite Deployment Planning Guide* .

In particular, pay attention to:

- OS hardening, turning off unused OS services (especially in Linux)
- OS minimization, using minimal OS packages

Installing Infrastructure Components

The following infrastructure components should be installed and secured prior to secure Messaging Server installation. You need to understand how all components in the infrastructure interconnect so that you can apply appropriate security measures to every interconnect.

- **Directory Server**: Messaging Server connects to the Oracle Directory Server Enterprise Edition, an LDAP-based directory server for user and group information and for provisioning. See the Directory Server Installation Scenario in *Unified Communications Suite Installation and Configuraiton Guide* and Oracle Directory Server Enterprise Edition Enhanced Security.
- **Communications Suite Directory Server Setup Script**: the `comm_dssetup.pl` script to prepare the Directory Server for Communications Suite installation.
- **GlassFish Message Queue**: is automatically installed with the other shared components. For the list of shared components that can be installed by the Communications Suite installer, run the `commpkg info --listPackages` command. To see the output for each supported operating system for more information, see the topic on shared components in *Unified Communications Suite Installation and Configuration Guide*. Additionally, see: Configuring and Managing Message Queue Security Services.
- **Messaging Server Sun Cluster HA Agent 7.0**: High Availability can be optionally installed.
- **DNS Server**: You must ensure that Domain Name System (DNS) is running and configured properly. For details, see the topic on DNS configuration in *Unified Communications Suite Installation and Configuration Guide*.
- **File System**: Recommended file systems for the message store are listed in the topic on message store file systems in *Messaging Server 8.0 Installation and Configuration Guide*.

In addition to dependent products, it is equally important to secure the other components within Unified Communications Suite for secure Messaging Server deployment. Review each component's Security Guide for security guidelines.

Performing a Messaging Server Initial Configuration

See *Messaging Server 8.0 Installation and Configuration Guide*.

When you perform a Messaging Server initial configuration by running the `configure` command, you are prompted for authentication credentials for the following:

- User Name and Group Name for Server Processes
- Directory Server manager (bind DN and password)
- Password for server administration

Post Installation Configuration

The high-level post-installation steps to configuring Messaging Server for a secure deployment include:

1. Installing Messaging Server Provisioning Tools
2. Enabling SMTP Relay Blocking
3. Enabling Startup After a Reboot
4. Configuring JMQ Notification
5. Configuring Certificate Based Authentication

For instructions, see the topic on Messaging Server initial configuration in *Messaging Server 8.0 Installation and Configuration Guide*.

> **Note**
> Once installation is complete, Oracle recommends encrypting and moving the initial state files and `configure.ldif` file, if generated.

Security Features

Messaging Server supports security features that enable you to keep messages from being intercepted, prevent intruders from impersonating your users or administrators, and permit only specific people access to specific parts of your messaging system. The Messaging Server security architecture is part of the security architecture of Oracle servers as a whole. It is built on industry standards and public protocols for maximum interoperability and consistency. For a general overview of Messaging Server security strategies, see the topic on planning Messaging Server security in *Unified Communications Suite Deployment Planning Guide*.

This section describes additional features, considerations, and recommendations for securing your Messaging Server deployment:

- Messaging Server Security Strategy for your Deployment
- MTA Security Guidelines
- ENS Security Guidelines
- Message Store Security Guidelines
- MMP Security Guidelines
- User Authentication Guidelines
- Message Encryption Guidelines

Messaging Server Security Strategy for your Deployment

Creating a security strategy is one of the most important steps in planning your deployment. Your strategy should meet your organization's security needs and provide a secure messaging environment without being overbearing to your users. For more information, see the topic on creating a security strategy in *Unified Communications Suite Deployment Planning Guide*.

How you set up the following topics impacts your security strategy guidelines:

- Identifying Password Policy Requirements
- Verifying File Ownership for Configuration Files
- Securely Monitoring and Auditing Your Messaging Server Deployment
- Tracking Security Patches
- Identifying Legal-intercept Requirements
- Securing Your Archiving Needs
- Disabling Users in Response to Abuse/Appeal Process
- Utilizing a Disk Consumption Growth Plan
- Preventing Unrelated Usage of Messaging Server Hosts and Virtual Machines
- Determining Security Capabilities of Your Supported Mail Clients

Identifying Password Policy Requirements

The user password login process is described in User Password Login for Messaging Server.

Review the following additional password policy recommendations:

1. Select a password policy system that meets your requirements and any additional requirements you might add at a later time.
2. Require your users to create high quality passwords on your site identity system's password change web page. Do not require your users to change passwords too frequently as it cause users to write their passwords on paper.
3. Directory Server has password policy capabilities, but if you enable password expiration, be sure the administrator and service accounts used by Messaging Server are exempt from expiration.
4. Keep a strong administration password.
5. Maintain administrative access policies for Messaging Server hosts.
6. Create Delegated Administrator access policies for domains.
7. If needed by Oracle Support, plan how to gather configuration files, excluding passwords. If, starting with Messaging Server 7 Update 5, you use Unified Configuration, this task is made much easier.

Verifying File Ownership for Configuration Files

Related topics in the both *Messaging Server System Administrator's Guide* and *Messaging Server Unified Configuration System Administrator's Guide* include:

- Check the Ownership of Critical Files
- User Mailbox Directory Problems

Securely Monitoring and Auditing Your Messaging Server Deployment

Monitoring your server is an important part of your security strategy. To identify attacks on your system, monitor message queue size, CPU utilization, disk availability, and network utilization. Unusual growth in the message queue size or reduced server response time may identify some of these attacks on MTA relays. Also, investigate unusual system load patterns and unusual connections. Review logs on a daily basis for any unusual activity.

Refer to Monitoring Messaging Server in Unified Configuration, which includes topics on:

- Automatic Monitoring and Restart
- Daily Monitoring Tasks
- Utilities and Tools for Monitoring
- Monitoring User Access to the Message Store
- Monitoring System Performance
- Monitoring Disk Space
- Monitoring the MTA
- Monitoring LDAP Directory Server
- Monitoring the Message Store
- Messaging Server Monitoring Framework Support
- SNMP Support

Additional guidelines for secure monitoring:

- Ensure you have the right monitoring and auditing tools for your specific deployment and that you have contingency plans in place.
- Enable MTA logging.

Tracking Security Patches

Be sure to install the most recent operating environment security patches and set up procedures to update the patches once every few months and in response to security alerts from the vendor. Be sure to pay close attention to NSS patches.

Identifying Legal-intercept Requirements

The following topics, available in *Messaging Server System Administrator's Guide* provide an overview for message archiving for legal and compliance purposes:

- Compliance Messaging Archiving
- imarchive with the AXS-One Archiving System
- Message Archiving Using the Sun Compliance and Content Management Solution (for legacy systems only)

Determine which Messaging Server capture mechanism is best to meet those requirements in your jurisdiction prior to responding to a compliance request.

Securing Your Archiving Needs

Once you have satisfied legal requirements, use your third-party archiving system in your jurisdiction so that it can be configured to delete messages from the archive (or make them unreadable by discarding encryption keys). Refer to Identifying Legal-intercept Requirements for message archiving options.

Disabling Users in Response to Abuse/Appeal Process

The following topics describe how to disable accounts, block mail to an account, and enable and disable services at different levels.

How Do I Temporarily Disable Mail Accounts?

You can disable services at the user level by setting the LDAP attribute `mailAllowedServiceAccess` to `none`. Mail will continue to be delivered but user access to the account over POP, IMAP, and HTTP will be disabled.

How Do I Block Email to a Specific Account?

See the `SEND_ACCESS` and `ORIG_SEND_ACCESS` mapping tables. Alternately, the user's account can be disabled by setting the `mailUserStatus` LDAP attribute to an appropriate value.

How Do I Enable and Disable Services?

See Enabling and Disabling Services.

Utilizing a Disk Consumption Growth Plan

Unusual disk consumption may identify some attacks on MTA relays.

See the following topics in *Messaging Server System Administrator's Guide*:

- Monitoring Disk Space
- Managing Message Store Partitions and Adding Storage

See the following topic in *Unified Communications Suite Deployment Planning Guide*:

- Performance Tuning Considerations for a Messaging Server Architecture

Preventing Unrelated Usage of Messaging Server Hosts and Virtual Machines

Oracle recommends that you do not use Messaging Server hosts or virtual machines for unrelated tasks. Single purpose hosts and virtual machines are better for securing your deployment. Be sure to turn off any unused Messaging Server services.

Determining Security Capabilities of Your Supported Mail Clients

For information on security and access control for mail clients and mail client infrastructure, refer to Security and Access Control where the following topics are covered:

- Authentication Mechanisms
- User Password Login
- Encryption and Certificate-based Authentication
- Mail Filtering and Access Control
- Client Access to POP, IMAP, and HTTP Services

Consider these questions when designing your Messaging Server security strategy:

- Do your mail clients support SMTP Authentication?
- Do your mail clients support standard SSL (STARTTLS on port `143` for IMAP, STLS on port `110` for POP, STARTTLS on port `587` for SMTP Submission)?
    - If not, do they support legacy non-standard SSL (IMAPS on port `993`, POPS on port `995`, SSL SMTP Submission on non-standard port `465`)?
- Do you have a plan in place to handle accidental/inappropriate blacklisting of your site by reputation services?

MTA Security Guidelines

Following secure guidelines protect your MTAs from unauthorized users, large quantities of spam, reduced response time, and used up disk space and resources. See *Unified Communications Suite Deployment Planning Guide* for general guidelines to protect your MTAs. This section provides additional details in the following topics:

- Securing Internal MTA Information
- Identifying, Integrating, and Configuring Anti-spam and Anti-virus Solutions with Messaging Server
- Securing ENS Server (7997) with Firewall and/or TCP Access Control Filters
- Creating a Narrow Scope of MTA Relay Blocking in INTERNAL_IP Mapping Table
- Using LMTP to Connect to Inbound MTAs and in Multi-tier Deployments
- Greylisting
- Forbidding Emailing Executable Code
- Throttling Incoming MTA Connections with MeterMaid
- Setting MTA Recipient Limits?
- Using Sieve Securely
- Using the MTA to Fix Messages from Bad Clients
- Configuring Secure ETRN Command Support

Securing Internal MTA Information

By default, Messaging Server enables four private commands called XADR, which returns information about how an address is routed internally by the MTA as well as general channel information, XCIR, which returns MTA circuit check information, XGEN, which returns status information about whether a compiled configuration and compiled character set are in use, and XSTA, which returns status information about the number of messages processed and currently in the MTA channel queues. Releasing such information may constitute a breach of security for some sites.

Sites might want to disable these commands for the `tcp_local` channel. For example, the commands to do so for Unified Configuration are as follows:

```
./msconfig set channel:tcp_local.options.disable_address 1
./msconfig set channel:tcp_local.options.disable_circuit 1
./msconfig set channel:tcp_local.options.disable_status 1
./msconfig set channel:tcp_local.options.disable_general 1
```

Options in the `.options` scope under channel are free form. As such, they are neither type-checked nor validated.

Identifying, Integrating, and Configuring Anti-spam and Anti-virus Solutions with Messaging Server

Refer to the following topics on anti-spam and anti-virus solutions:

- Integrating Spam and Virus Filtering Programs Into Messaging Server in Unified Configuration
- "About the Milter Plugin," in *Messaging Server System Administrator's Guide*
- "Handling Forged Email by Using the Sender Policy Framework" in *Messaging Server System Administrator's Guide*
- Protecting Against Spammers who Compromise Messaging Server User Accounts
- Performance Tuning DNS Realtime BlockLists (RBL) Lookups

In addition, consider these guidelines:

- Make sure your domain's MX records point directly to Messaging Server's MTA and have the Messaging Server call out to anti-spam/anti-virus systems preferably through the spam plug-in or Milter mechanism.
- Filter both inbound and outbound mail.
- Consider restricting outbound port 25 to outbound MTAs only.

Creating a Narrow Scope of MTA Relay Blocking in INTERNAL_IP Mapping Table

To use the INTERNAL_IP mapping table for MTA Relay Blocking, refer to the following topics in either *Messaging Server System Administrator's Guide* or *Messaging Server Unified Configuration System Administrator's Guide*:

- Preventing Mail Relay
- Configuring SMTP Relay Blocking

Starting with Messaging Server 7 Update 5, you can accomplish STMP relay blocking during initial configuration. The `configure` command prompts for a list of IP addresses of internal systems that are allowed to relay. If the list of such systems is empty, then the default settings from the `configure` command are used.

Using LMTP to Connect to Inbound MTAs and in Multi-tier Deployments

LMTP can provide an extra layer of security between the relays and the back end message stores. See the following topics:

- Using LMTP to Connect to Inbound MTAs in *Messaging Server System Administrator's Guide*
- http://msg.wikidoc.info/index.php/Multi-host_defragmentation_channel_operation
- Messaging Server NFS Guidelines and Requirements
- Implementing Local Message Transfer Protocol (LMTP) for Messaging Server in *Messaging Server System Administrator's Guide*

Greylisting

See: Protecting Against Spammers who Compromise Messaging Server User Accounts

Forbidding Emailing Executable Code

Use the conversion channel to replace the MIME contents of messages, that is, to replace or remove image or executable files based on file name extensions

For example, to remove `*.gif` and `*.exe` files, use the following steps.

> **Note**
> This example was tested on Messaging Server 6.2 running patch 118207-63.

1. Enable the conversion channel for emails being delivered locally (through the `ims-ms` channel) by adding the following to the *msg_base*`/config/mappings` file:

```
CONVERSIONS

! Convert all emails coming in to local users from outside
  IN-CHAN=tcp_*;OUT-CHAN=ims-ms;CONVERT Yes
```

2. Write conversion script commands for each attachment type you want to remove by creating the *msg_base*`/config/conversions` file and adding the following:

```
! Delete *.gif attachments by filename
in-channel=*; in-type=*; in-subtype=*;
  in-dparameter-name-0=filename; in-dparameter-value-0=*.gif;
  DELETE=1

! Delete GIF attachments by media type
in-channel=*; in-type=image; in-subtype=gif;
  DELETE=1

! Delete *.exe attachments
in-channel=*; in-type=*; in-subtype=*;
  in-dparameter-name-0=filename; in-dparameter-value-0=*.exe;
  DELETE=1
```

   These example entries show deletion based on file extension and on media type name. Several entries are needed to catch all of the variants you want deleted.

3. Enable the rule by rebuilding the MTA configuration and restarting.

```
cd <msg_base>/sbin
./imsimta cnbuild
./imsimta restart
```

Throttling Incoming MTA Connections with MeterMaid

MeterMaid is a server that can provide centralized metering and management of connections and transactions, including through monitoring IP addresses SMTP envelope addresses. Functionally,

MeterMaid can be used to limit how often a particular IP address can connect to the MTA. Limiting connections by particular IP addresses is useful for preventing excessive connections used in denial-of-service attacks.

See the following topic:

- Using and Configuring MeterMaid for Access Control

Setting MTA Recipient Limits?

See: http://msg.wikidoc.info/index.php/Channel_configuration

Using Sieve Securely

Review your `MAX_*` options settings relevant to Sieve filter limits, especially `MAX_NOTIFYS`.

> **Note**
> Notify, Forward, and Redirect can potentially increase the load of generating new messages. You need to consider if abusers could exploit such features by generating message loops or exponential growth of messages.

For Sieve external lists, enable setup carefully only allowing specific criteria. Some Sieve filter user education/Sieve filter creation interface guidelines to consider:

- Discourage users from attempting to personally block spam by using Sieve.
- Check that the interface generates efficient Sieves (for example: lists, wildcard matches, and so on)

Review:

- Sieve Filter Support

Using the MTA to Fix Messages from Bad Clients

If users use email clients that especially vulnerable to buffer overruns, malicious embedding in malformed header lines, and so on, consider configuring the MTA with maximal MTA MIME processing and fixing up messages passing through the MTA with the `inner` MTA channel option.

Configuring Secure ETRN Command Support

Consider explicitly configuring the `ETRN` commands that the MTA honors. See the `ETRN_ACCESS` mapping table, the `*etrn` channel options, and the `ALLOW_ETRNS_PER_SESSION` TCP/IP channel option.

Review to the following topics:

- ETRN_ACCESS mapping table
- ETRN Command Support

ENS Security Guidelines

Securing ENS Server (7997) with Firewall and/or TCP Access Control Filters

To deploy the Event Notification Service (ENS) with Messaging Server, see: *Unified Communications Suite Event Notification Service Guide*.

Note
The current implementation of ENS does not provide security on events that can be subscribed to. Thus, a user could register for all events, and portions of all other users' mail. Because of this it is strongly recommended that the ENS subscriber be on the "safe" side of the firewall at the very least.

A firewall system generally controls what TCP/IP communications are allowed between internal networks and the external world. Firewalls prevent packets considered to be unsafe from passing through.

Message Store Security Guidelines

The most important data in the Messaging Server is the data in the message store. Physical access and root access to the message store must be protected. See the topic on protecting your message store in *Unified Communication Suite Deployment Planning Guide* for general guidelines to protect your message store. In addition, you should review Message Store Management. This section provides additional details in the following topics:

- Securing Your Backup System
- Using configutil Parameters for Securing Messaging Server
- Being Aware of IMAP ACLs
- Disabling IMAP Shared Folders if Not Needed

Securing Your Backup System

The process for backing up the Messaging Server is described in "Message Store Backup and Restore" in *Messaging Server System Administrator's Guide*.

Some security guidelines to consider for message store backup:

- Be sure that such a system does not leave unneeded data.
- Backup systems that encrypt data improve your security if you manage the encryption keys properly.

Using configutil Parameters for Securing Messaging Server

The `service.feedback.notspam`, `service.feedback.spam`, and `service.http.ipsecurity` `configutil` parameters are used to secure Messaging Server.

Being Aware of IMAP ACLs

See the following topics: The message store `readership` command-line utility and "Managing Shared Folders" in *Messaging Server System Administrator's Guide*.

Disabling IMAP Shared Folders if Not Needed

Disable shared folders if not in use. See: "Disabling IMAP Shared Folders" in *Messaging Server System Administrator's Guide*.

MMP Security Guidelines

The MMP serves as a proxy for the message store, therefore, it needs to protect access to end user data and guard against unauthorized access. See the topic on protecting MMPs in *Unified Communications Suite Deployment Planning Guide* for general guidelines.

You can use the server machine on which the multiplexor is installed as a firewall machine. By routing all

client connections through this machine, you can restrict access to the internal message store machines by outside computers. The multiplexors support both unencrypted and encrypted communications with clients.

For information on MMP badguy throttling and MMP connection limits, see: MMP Reference

User Authentication Guidelines

User authentication allows end users to securely log in through their mail clients to retrieve their mail messages. See the topic on planning Messaging user authentication in *Unified Communications Suite Deployment Planning Guide* for general guidelines. This section adds the following topics:

- Acquiring SSL Server Certificates for the Server Domains
- Requiring SMTP Authentication for Mail Submission

Acquiring SSL Server Certificates for the Server Domains

Refer to *Unified Communications Suite Certificate Authentication Guide* for more information on certificate based authentication.

Note the following recommendations:

- Acquire SSL server certificates for server domains to which your users will connect from a third-party CA. If you also wish to secure inter-deployment connections (recommended for a geographically distributed deployment as well as helpful to meet legal requirements in some jurisdictions), also get certificates for your Directory Servers and back-end IMAP/POP storage servers.
- Purchasing Certificate Authority (CA) service or software for your enterprise may be cost effective if you have many hosts in your deployment. Be sure to use at least 2048-bit RSA with SHA 256 signatures per current guidelines unless your jurisdiction does not permit that or some of your mail clients do not support that.
- The `certutil` provided with Solaris and Communications Suite Installer too can be used with the -g 2048 and -Z sha256 switches). Once enabled, you can configure SSL.

Some guidelines for SSL include:

- Having a plan for SSL certificate or CA expiration.
- Turning on SSL where required (external services, possible internal services)
- Requiring SSL where possible (`RestrictPlainPasswords`, `plaintextmincipher`)

Requiring SMTP Authentication for Mail Submission

SMTP Authentication, or SMTP Auth (RFC 2554) is the preferred method of providing SMTP relay server security. SMTP Auth allows only authenticated users to send mail through the MTA.

- Using Authenticated Addresses from SMTP AUTH in Header
- SMTP Password Login
- authrewrite channel option

Message Encryption Guidelines

The topic on planning message encryption strategies in *Unified Communications Suite Deployment Planning Guide* covers S/MIME and Encryption with SSL for encryption and privacy solutions. Review the following guidelines and recommendations:

- Determining SSL Cipher Suites
- Using Solaris Crypto Framework in Place of NSS Default Software Token

Determining SSL Cipher Suites

See: Configuring Encryption and Certificate-Based Authentication

Review your SSL Cipher framework to determine the following:

- Do your mail clients work if the mail server only supports modern AES cipher suites, modern AES and slow-but-secure 3DES? Or, are legacy RC4 cipher suites required?
- Is SSL3 required, or can you restrict to TLS?

Using Solaris Crypto Framework in Place of NSS Default Software Token

If you use SSL for encryption, you can improve server performance by installing a hardware encryption accelerator.

Security Considerations for Developers

For secure programming best practices, refer to *Messaging Server MTA Developer's Reference*.

Messaging Server NFS Guidelines and Requirements

Oracle Communications Messaging Server NFS Guidelines and Requirements

Follow these NFS guidelines and requirements for use with Messaging Server:

- Time synchronization: Keeping synchronized time across a Messaging Server deployment is necessary to map events from one network component to events on another component. Failure to keep system clocks synchronized can result in all sorts of strange errors. Configure Network Time Protocol (NTP) to ensure that time is synchronized across your deployment.
  For more information, see the following articles, available from the Internet Archive:
  - Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP
  - Using NTP to Control and Synchronize System Clocks - Part II: Basic NTP Administration and Architecture
  - Using NTP to Control and Synchronize System Clocks - Part III: NTP Monitoring and Troubleshooting
- Where to use NFS: You can use NFS on MTA relay machines, for LMTP, for autoreply histories, and for message defragmentation. In addition, NFS can be supported on BSD-style mailboxes (`/var/mail/`) as well as for message stores.
- For instructions on how to configure NetApp storage appliances (called filers) with the Messaging Server message store, see Using NetApp Filers with Messaging Server Message Store.

> **Note**
> The ability to place the message store on an NFS server is not intended to allow other NFS clients to access that store, but only to take advantage of the configuration flexibility afforded by NFS servers. No other process from other systems, other than the message store server, should be accessing the message store.

Chapter 58. Setting Up a No Phishing Zone

Messaging Server: Setting Up a No Phishing Zone

> **Update**
> See Protecting Against Spammers who Compromise Messaging Server User Accounts for best practices on combating this issue.

Experienced Messaging Server administrators know that dealing with spam is a high-priority job requiring constant attention as spammers evolve and refine their methods of attacks. Recently, many admins have noted the rise of phishing attacks, especially against (but not exclusively) webmail clients.

Long time Messaging Server admins have been exchanging ideas and collaborating on all aspects of Messaging Server, including anti-spam/anti-virus techniques, by using the Info-IMS@arnold.com forum. (In brief, this alias is the independent discussion forum for those interested in Messaging Server and all its permutations (Java Enterprise System, Sun ONE, iPlanet Messaging Server). If you are a Messaging Server administrator and haven't yet subscribed to this alias, we highly recommend that you do so, here.

An email thread from July 2008 highlighted the phishing problem, especially in the EDU space. Many ideas were suggested on how to combat this particular spam issue. You can view the full thread on this topic at the following URL:

http://lists.balius.com/pipermail/info-ims-archive/2008-July/029647.html

Following is a summary of anti-spam techniques to consider:

- Examine the sent folder to get the source IP of the submission then "null route" the IP address on the Webmail front ends.
- Configure MeterMaid, which shipped with Messaging Server 6.3. MeterMaid limits the number of messages a user can send in a number of minutes regardless of source (SMTP, Webmail). More info on configuring MeterMaid here.
- Use the `./imsconnutil -k -u` *uid* command to disconnect the offending user account.
- Block the offending IP address at your firewall.
- Set the `inetuserstatus` attribute for the offending user to **inactive**, change the user's password, then clear the queue(s), though this technique is in response to an attack, rather than preventing or detecting the attack.
- Enable the Directory Server audit log. Monitor for changes to directory entries, such as signature files and reply-to addresses, by using a script and crontab to classify likely compromised accounts.
- Read the recommendation for how to deploy the Messaging Server MTA and anti-spam/anti-virus scanning systems:
  https://blogs.oracle.com/factotum/entry/messaging_server_mta_preferred_practices
- Call out to MeterMaid from the FROM_ACCESS mapping table passing as data the user authentication, rather than (or perhaps in additon to) calling out to MeterMaid from PORT_ACCESS mapping table passing as data the source IP. This technique limits how many messages some (authenticated) user can submit.
- Use Postfix/Policyd. Then change the default smtphost of Webmail to use it.
- Use this list of list of these password phishing reply addresses:
  http://code.google.com/p/anti-phishing-email-reply/
- Implement scanning systems on both incoming and outgoing email.
- Use the http://www.senderbase.org/ database.
- Starting with Messaging Server 7 Update 2, you can use LOG_ACTION to block submissions of local senders who might be sending spam.

Chapter 59. Using IPv6 with Messaging Server

Using IPv6 with Oracle Communications Messaging Server

This information provides an overview of IPv6 with Messaging Server. It does not discuss how to configure your operating system or network for IPv6 operation. It is assumed that at a minimum you have configured the host operating system to have IPv6 network interfaces available for use by Messaging Server. The host operating system should have both IPv4 and IPv6 network interfaces enabled so that Messaging Server can communicate with IPv4-only clients and servers. Furthermore, operation in a pure IPv6 only environment is not supported by Messaging Server.

> **Note**
> In reading this document, it is important to understand that IPv6 addresses are stored in the DNS using entries called "AAAA" records, often pronounced "quad A". DNS "A" records are, of course, the DNS entries for IPv4 addresses. In this document, we will use the phrase "address records" to collectively refer to both A and AAAA DNS records.

Topics:

Basic Configuration

By default, Messaging Server only has IPv4 support enabled. Unless you change one of the configuration options described below, Messaging Server only accepts inbound IPv4 connections and only initiates outbound IPv4 connections. When you enable use of IPv6 for either inbound or outbound connections, Messaging Server continues to also allow IPv4 inbound connections and to use IPv4 outbound connections for host names which resolve to DNS A records.

The configuration options to control Messaging Server's use of IPv6 are `local.ipv6.in`, `local.ipv6.out`, and `local.ipv6.sortorder`. They are configured with the `configutil` utility.

The following table shows the `configutil` parameters and their descriptions:

IPv6 Configuration Parameters

Parameter (possible values)	Description
`local.ipv6.in` (0 or 1)	Allow (1) or disallow (0) inbound IPv6 connections. When set to the value 1, all services allow inbound IPv6 and IPv4 connections (for example, POP, IMAP, SMTP, LMTP, HTTP, and so on). When set to the value 0, only IPv4 inbound connections are permitted. The default value for this option is 0. Note: The behavior of the SNMP subagent is not altered by this option. The behavior of the platform's SNMP services as well as any SNMP subagents is controlled through the platform's SNMP configuration system.
`local.ipv6.out` (0 or 1)	Allow (1) or disallow (0) outbound IPv6 connections. When this option is set to the value 1, all services which initiate outbound connections may use either IPv6 or IPv4 (for example, SMTP, LMTP, LDAP, remote POP and IMAP mail collection by mshttpd, and so on). The choice of IPv4 versus IPv6 may further depend on a specific service's configuration. See 3. Outbound IPv6 Connections for further discussion of this topic.
`local.ipv6.sortorder` (DEFAULT, A-AAAA, AAAA-A, A, AAAA)	When `local.ipv6.out` is set to the value 1, Messaging Server will request both A and AAAA records from the DNS when performing DNS-based host name resolution. These records will be returned in whatever order the DNS sees fit. Further re-ordering may be performed by the platform's `getaddrinfo()` implementation. On Oracle Solaris, `getaddrinfo()` performs the re-ordering algorithm described in RFC 3484. See `getaddrinfo(3SOCKET)` for details. If you wish Messaging Server to attempt all A records before any AAAA records, then set this option to have the value `A-AAAA`. To attempt all AAAA records first, use `AAAA-A`. To only honor AAAA records and thus only make IPv6 outbound connections, use the value `AAAA`. To only honor A records and thus only make IPv4 outbound connections, use `A`. To attempt the records in the order returned by `getaddrinfo()`, use the value `DEFAULT`. The default setting for this option is `DEFAULT`.

Outbound IPv6 Connections

1. To enable outbound IPv6 connections for all services, set `local.ipv6.out` to 1:

```
# configutil -o local.ipv6.out -v 1
```

2. If using a compiled configuration for the MTA, run the following command:

```
# imsimta cnbuild
```

It is not possible to enable IPv6 for specific services, it can only be enabled for all Messaging Server services.

When services attempt to connect to a remote host using a host name (as opposed to an IP address), they will first resolve the host name to one or more address records. The nature of the resulting address

records will then govern whether or not an outbound IPv4 or IPv6 connection is used. For an A record, an IPv4 connection is used. For an AAAA record, an IPv6 connection is used. The `local.ipv6.sortorder` option may be used to influence which sort of records are used preferentially.

Services configured to use an IP address rather than a host name will establish a connection whose type--IPv4 versus IPv6--matches that of the configured IP address. An IPv4 connection for an IP address will have the form `a.b.c.d`, and an IPv6 connection will have the form `a:b:c:d:e:f:g:h`. For example, a tcp_ channel configured with `daemon 10.1.110.6` will only attempt IPv4 outbound connections.

Inbound IPv6 Connections

- To enable inbound IPv6 connections for all services, set `local.ipv6.in` to 1:

```
# configutil -o local.ipv6.in -v 1
```

As with inbound connections, it is not possible to enable IPv6 for specific services. They can only be enabled for all Messaging Server services.

When inbound IPv6 connections are permitted on Oracle Solaris, all inbound connections appear to be IPv6 connections, even when the remote client is using IPv4. Specifically, when a remote client initiates an IPv4 connection to Messaging Server and Messaging Server is accepting both IPv4 and IPv6 inbound connections, then Oracle Solaris will report the remote source IPv4 address `a.b.c.d` as the IPv6 address `::ffff:a.b.c.d`. Messaging Server will automatically map that IPv6 address back to the IPv4 address `a.b.c.d` before logging it or presenting it to any access tests.

TCP wrappers and MMP's Bad Guys Lists

The TCP wrappers used by `service.`**`.domainallowed`** and **`service.`**`.domainnotallowed` as well as the MMP's Bad Guys lists have been updated to allow specification of IPv6 addresses and IPv6 CIDRs (Classless Inter-Domain Routing). Both should be specified within square brackets. For an IPv6 address,

```
[a:b:c:d:e:f:g:h]
```

and for an IPv6 CIDR,

```
[a:b:c:d:e:f:g:h/p]
```

where `a:b:c:d:e:f:g:h` is the IPv6 address and p is the routing prefix. For IPv6, the routing prefix is an integer between 0 and 128, inclusive. IPv6 "`::`" short hand notation is supported.

When IPv6 support was added, the TCP wrappers and Bad Guys facilities were updated to allow IPv4 CIDRs in addition to IPv4 netmasks. Consequently, both the IPv4 CIDR format,

```
a.b.c.d/p
```

as well as the IPv4 netmask format

```
a.b.c.d\|w.x.y.z
```

are supported.

Connection Counters

The connection counters controlled with the `service.*.connlimits` options have been updated to allow specification of IPv6 addresses and IPv6 CIDRs. For an IPv6 address,

```
[a:b:c:d:e:f:g:h]:m
```

and for an IPv6 CIDR,

```
[a:b:c:d:e:f:g:h/p]:m
```

where `a:b:c:d:e:f:g:h` is the IPv6 address, `p` is the routing prefix, and `m` is the connection limit.

When IPv6 support was added, the Connection Counter facility was updated to allow IPv4 CIDRs in addition to IPv4 netmasks. Consequently, both the IPv4 CIDR format,

```
a.b.c.d/p:m
```

as well as the IPv4 netmask format,

```
a.b.c.d\|w.x.y.z:m
```

are accepted.

Formerly, the Connection Counter facility used the format

```
0.0.0.0\|0.0.0.0:m
```

to specify a default limit for IPv4 connections which did not match a more specific limit. The Connection Counter facility has been updated to have a default limit for IPv6 connections as well as a more general default for connections which are either IPv4 or IPv6. This generic limit is specified using the format

```
:m
```

And, the default limit for IPv6 connections has the format

```
[::0/0]:m
```

The precedence for the connection limits are then

1. Highest precedence to specific patterns of the forms `a.b.c.d/m`, `a.b.c.d|w.x.y.z:m`, and `[a:b:c:d:e:f:g:h]:m`.
2. IPv4 connections which don't match any patterns from 1, are then limited by `0.0.0.0|0.0.0.0:m` (or, equivalently, `0.0.0.0/0:m`), if it is specified and `:m` otherwise.
3. IPv6 connections which don't match any patterns from 1, are limited by y `[::0/0]:m` if it is specified and `:m` otherwise.

Mapping Tables

The MTA mapping tables have had support for IPv6 addressing for many years; no new functionality to support IPv6 needed to be added. The basic format for specifying an IPv6 address or IPv6 CIDR in the mapping tables is:

```
${ipv6}
```

where `ipv6` is the IPv6 address or IPv6 CIDR. Refer to the topic on controlling access with mapping tables for further details.

Caveats

Messaging Server still uses its own internal resolver library to resolve MX and TXT DNS records. This is done because Oracle Solaris and other operating systems do not provide thread-safe resolver libraries for looking up MX or TXT records. (The `getXbyY()` routines are thread-safe but do not support MX or TXT record lookup operations.) Messaging Server's library for resolving MX and TXT records does not support the use of IPv6 for communicating with DNS servers.

MTA systems that need to do MX and TXT record lookups must be able to query DNS servers by using IPv4 network connections.

If a destination domain has an MX record(s) in DNS, the FQDN for a mail exchanger can be resolved to an IP address by using either A records, AAAA records, or both. The `local.ipv6.sortorder` `configutil` configuration parameter determines which address has precedence.

If a destination domain does not have an MX record in DNS, the domain name itself is used as the FQDN which is then resolved to an IPv4 and/or IPv6 address as previously described.

At present, Messaging Server does not provide a mechanism to bind services to specific IPv6 interfaces. Any site requiring this support should contact Oracle. This feature is absent because IPv6 doesn't provide for it; the designers of IPv6 disapproved of binding services to specific IP addresses and thus left the concept out of IPv6. However, most operating systems provide mechanisms for binding a service to a specific IPv6 interface. Unfortunately, these mechanisms are non-standard and, on some platforms, have changed with OS versions.

Use of JMQ is not supported when IPv6 is used; ENS should be used instead in that case.

Chapter 60. Using NetApp Filers with Messaging Server Message Store

Using NetApp Filers with Messaging Server Message Store

This technical notes describes how to configure NetApp storage appliances called **filers** with the Messaging Server message store.

> These instructions apply starting with **Messaging Server 6.3**.

This technical note contain the following sections:

About This Technical Note

The Messaging Server message store contains the user mailboxes for a particular Messaging Server instance. The size of the message store increases as the number of mailboxes, folders, and log files increase.

As you add more users to your system, your disk storage requirements increase. Depending on the number of users your server supports, the message store might require one physical disk or multiple physical disks. Messaging Server enables you an add more stores as needed.

One approach to adding more stores is by using **storage appliances**. NetApp storage appliances called filers integrate seamlessly with Messaging Server in the message delivery environment. Filers are reliable and provide excellent performance, scalability, and data availability. Filers provide high-performance access to a single copy of the data, which is shared across all types of UNIX®clients through NFS.

The high-level steps to configure the NetApp filer for Messaging Server are:

1. Planning disk capacity
2. Creating volumes
3. Configuring Messaging Server to access the NetApp filer

In addition, you can use Snapshot™to create periodic copies for data protection in the event of server failure or loss of data. You can use SnapRestore® to quickly restore mailboxes from the snapshots taken previously. You can dump the Snapshot copies to tape library using NDMP and store them offsite. For more streamlined disaster recovery (DR) purposes, you can send these Snapshot copies by using SnapMirror® to a NetApp NearStore® system located at a secondary site or data center.

Planning Disk Capacity and Creating Volumes

You need to create a volume (or volumes) on the filer before installing Messaging Server. To avoid disk I/O bottlenecks, configure the system with as many spindles as possible. Note that more spindles in a volume means longer RAID reconstruction time in case disk failure happens.

Note
The message store file system on the NetApp filer can only be mounted by one Messaging Server host. Sharing the same message store file system by more than one Messaging Server is not supported.

To Create a Volume on a NetApp Data ONTAP 7G (Flexible Volume)

The following commands create an aggregate (`aggr1`) and a flexible volume (`eng`).

- To create an aggregate called `aggr1` with 10 spindles (disks):

```
aggr create aggr1 10
```

- To create a 20 GB flexible volume called `eng`:

```
vol create eng aggr1 20g
```

To Create a Volume on a NetApp Data ONTAP 6.5 and Older (Traditional Volume)

1. The following command creates a volume called `eng` with 10 spindles:

```
vol create eng 10
```

2. Export the volume(s) to Messaging Server through NFS.
   Add the following entry to the system `/etc/exports` file on the filer (the server is `msg1`).

```
/vol/eng -root=msg1
```

3. Run the Data ONTAP exportfs command.

```
exportfs -a
```

4. Mount the volume `eng` from server `msg1`.

```
mount filer:/vol/eng /eng
```

5. Use `/eng` as the path to the message store.

Configuring Messaging Server to Work with NetApp Filers

After creating the volume, you need to configure Messaging Server so that it can function with the NetApp device.

To Configure Messaging Server to Work with NetApp Filers

1. Configure the temporary database directory on the Message Store host by setting the `store.dbtmpdir` parameter to a directory under `/tmp`.
   For example:

   ```
   configutil -o store.dbtmpdir -v /tmp/mboxlist
   ```

2. Configure the lock directory on the Messaging Server host by setting the `local.lockdir` parameter to a directory under `/tmp`.
   For example:

   ```
   configutil -o local.lockdir -v /tmp/lockdir
   ```

   Note
   Do not use the same directory for both `store.dbtmpdir` and `local.lockdir`.

3. Configure the default log directory by setting the `logfile.default.logdir` to a directory on local storage. This prevents the monitoring tools from hanging when the remote file system is unavailable. Make sure the local `logdir` is writable by the Messaging Server user.
   For example:

   ```
   configutil -o logfile.default.logdir -v /path/to/logdir
   ```

Chapter 61. Veritas Cluster Server Agent Installation

Veritas Cluster Server Agent Installation

Messaging Server can be configured with Veritas Cluster Server 3.5, 4.0, 4.1, and 5.0. Be sure to review the Veritas Cluster Server documentation prior to following these procedures. Veritas cluster Server agent for Messaging Server is part of the Messaging Server 7.0 core package and is installed during Messaging Server installation only.

This document contains the following sections:

Veritas Cluster Server Requirements

Veritas Cluster Software is already installed and configured as described in the following instructions along with the Messaging Server software on both nodes.

VCS Installation and Configuration Notes

The following instructions describe how to configure Messaging Server as an HA service, by using Veritas Cluster Server. The default `main.cf` configuration file sets up a resource group called `ClusterService` that launches the VCSweb application. This group includes network logical host IP resources like `csgnic` and `webip`. In addition, the `ntfr` resource is created for event notification.

To Configure Messaging Server as an HA Service by Using Veritas Cluster Server

These Veritas Cluster Server instructions assume you are using the graphical user interface to configure Messaging Server as an HA service.

1. Launch Cluster Explorer from one of the nodes.
   To launch Cluster Explorer, run the following command:

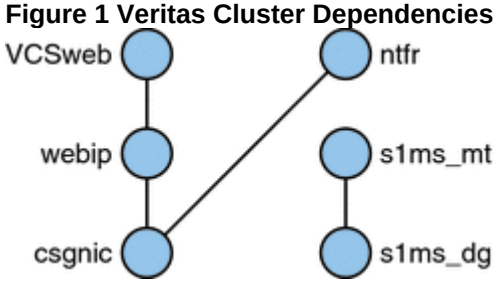
   ```
   /opt/VRTSvcs/bin/hagui
   ```

   The `VRTScscm` package must be installed to use the GUI.

2. Using the Cluster Explorer, add a service group called `MAIL-RG`.
3. Add `s1ms_dg` disk group resource of type `DiskGroup` to the service group `MAIL-RG` and enable it.
4. Add `s1ms_mt` mount resource of type `Mount` to the service group `MAIL-RG`.
   Click the Link button to enable linking resources, if they are not already enabled.
5. Create a link between `s1ms_mt` and `s1ms_dg`.
6. Enable the resource `s1ms_mt`.

The following figure depicts the dependency tree:

Figure 1 Veritas Cluster Dependencies



7. Run the Communications Suite Installer to install the Messaging Server software.
   a. Run the Messaging Server Initial Runtime Configuration (`configure`) from the primary node (for example, `Node_A`) to configure Messaging Server. The initial runtime configuration program asks for the Fully Qualified Host Name. Enter the logical hostname. The program also asks to specify a configuration directory. Enter mount point of the file system related to shared disk.
   b. Messaging Server running on a server requires that the correct IP address binds it. This is required for proper configuration of Messaging in an HA environment. Execute `ha_ip_config` command to bind to correct IP address.

      ```
      <msg-svr-base>/sbin/ha_ip_config
      ```

      The `ha_ip_config` program asks for the Logical IP address and Messaging Server Base (*msg-svr-base*).

   c. During Messaging Server installation, VCS agent related directory `vcsha` is created under the Messaging Server base directory, which will have VCS HA agent related files. Run `config-vcsha` to copy agent files to VCS configuration.

      ```
      <msg-svr-base>/sbin/config-vcsha
      ```

      Messaging Server and the Veritas agent are available on `Node_A`.

8. Switch to the backup node (for example, `Node_B`).
9. Run the Communications Suite Installer to install Messaging Server software on the backup node (`Node_B`).
10. After installing Messaging Server, use the `useconfig` utility to obviate the need for creating an additional initial runtime configuration on the backup node (`Node_B`).
    The `useconfig` utility enables you to share a single configuration between multiple nodes in an HA environment. This utility is not meant to upgrade or update an existing configuration. To enable the utility, run `useconfig` to point to your previous Messaging Server configuration:

    ```
    <msg-svr-base>/sbin/useconfig
    msg-svr-base/data/setup/configure_YYYYMMDDHHMMSS
    ```

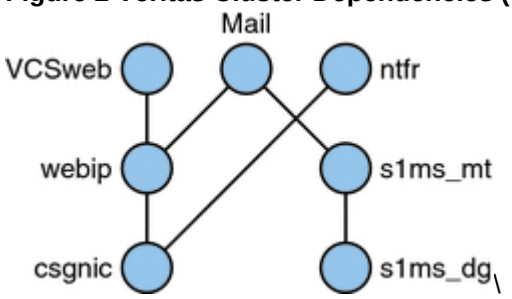
    where `configure_`*YYYYMMDDHHMMSS* is your previous configuration settings file.

11. As VCS HA agent is part of Messaging Server installation, run `config-vcsha` to copy agent files to VCS configuration.

```
<msg-svr-base>/sbin/config-vcsha
```

The Veritas agent is also now installed on `Node_B`.

12. From the Veritas Cluster Server Cluster Manager, Select ImportTypes from the File menu, which will display a file selection box.
13. Import the `MsgSrvTypes.cf` file from the `/etc/VRTSvcs/conf/config` directory.
14. Import this type file.
    You need to be on a cluster node to find this file.
15. Create a resource of type `MsgSrv` (for example, `Mail`).
    This resource requires the logical host name property to be set.
16. The Mail resource depends on `s1ms_mt` and `webip`. Create links between the resources as shown in the following dependency tree:
    Figure 2 Veritas Cluster Dependencies (s1ms_mt and webip)



    a. Enable all resources and bring `Mail` online.
    b. All servers should be started. Switch over to `Node_A` and check if the High Availability configuration is working.

MsgSrv Attributes and Arguments

This section describes MsgSrv additional attributes and arguments that govern the behavior of the mail resource.

Table 1 Veritas Cluster Server Attributes

Attribute	Description
FaultOnMonitorTimeouts	If unset (=0), monitor (probe) time outs are not treated as resource fault. Recommend setting this to 2. If the monitor times out twice, the resource will be restarted or failed over.
ConfInterval	Time interval over which faults/restarts are counted. Previous history is erased if the service remains online for this duration. Suggest 600 seconds.
ToleranceLimit	Number of times the monitor should returnOFFLINE for declaring the resource FAULTED. Recommend leaving this value at 0 (default).

Table 2 MsgSrv Arguments

Parameter	Description
State	Indicates if the service is online or not in this system. This value is not changeable by the user.
LogHostName	The logical host name that is associated with this instance.
PrtStatus	If set to TRUE, the online status is printed to the Veritas Cluster Server log file.
DebugMode	If set to TRUE, the debugging information is sent to the Veritas Cluster Server log file.

To obtain the current values of following debug options:

```
# pwd
/opt/VRTSvcs/bin

./hares -value ms-srvr DebugMode
./hares -value ms-srvr PrtStatus
```

To set the following debug options:

```
# pwd
/opt/VRTSvcs/bin

./hares -modify ms-srvr PrtStatus true
./hares -modify ms-srvr DebugMode true
```

Unconfiguring High Availability

This section describes how to unconfigure high availability. To uninstall high availability, follow the instructions in your Veritas or Sun Cluster documentation. The High Availability unconfiguration instructions differ depending on whether you are removing Veritas Cluster Server or Sun Cluster.

To Unconfigure the Veritas Cluster Server

This section describes how to unconfigure the high availability components for Veritas Cluster Server.

1. Bring the `MAIL-RG` service group offline and disable its resources.
2. Remove the dependencies between the `mail` resource, the `logical_IP` resource, and the `mountshared` resource.
3. Bring the `MAIL-RG` service group back online so the `sharedg` resource is available.
4. Delete all of the Veritas Cluster Server resources created during installation.
5. Stop the Veritas Cluster Server and remove following files on both nodes:

```
/etc/VRTSvcs/conf/config/MsgSrvTypes.cf
/opt/VRTSvcs/bin/MsgSrv/online
/opt/VRTSvcs/bin/MsgSrv/offline
/opt/VRTSvcs/bin/MsgSrv/clean
/opt/VRTSvcs/bin/MsgSrv/monitor
/opt/VRTSvcs/bin/MsgSrv/sub.pl
```

6. Remove the Messaging Server entries from the `/etc/VRTSvcs/conf/config/main.cf` file on both nodes.
7. Remove the `/opt/VRTSvcs/bin/MsgSrv/` directory from both nodes.

Chapter 62. Using and Configuring MeterMaid for Access Control

Using and Configuring MeterMaid for Access Control

MeterMaid is a server that can provide centralized metering and management of connections and transactions through monitoring IP addresses and SMTP envelope addresses. Functionally, MeterMaid can be used to limit how often a particular IP address can connect to the MTA. Limiting connections by particular IP addresses is useful for preventing excessive connections used in denial-of-service attacks. MeterMaid supplants `conn_throttle.so` by providing similar functionality, but extending it across the Messaging Server installation. No new enhancements are planned for `conn_throttle.so` and MeterMaid is its more effective replacement.

Topics:

Technical Overview

`conn_throttle.so` is a shared library used as a callout from the MTA's mapping table that uses an in-memory table of incoming connections to determine when a particular IP address has recently connected too often and should be turned away for awhile. While having an in-memory table is good for performance, its largest cost is that each individual process on each server maintains its own table.

In most cases, the `conn_throttle.so` callout is done in the `PORT_ACCESS` mapping that is accessed by the Dispatcher, a single process on each system. The only cost is that there is a separate table per server.

The primary improvement by MeterMaid is that it maintains a single repository of the throttling information that can be accessed by all systems and processes within the Messaging Server environment. It continues to maintain an in-memory database to store this data to maximize performance. Restarting MeterMaid will lose all information previously stored, but since the data is typically very short lived, the cost of such a restart (done infrequently) is very low.

Theory of Operations

MeterMaid's configuration is maintained by the `msconfig` command or the `configutil` command in legacy configuration.

MeterMaid is accessed from the MTA through a mapping table callout using `check_metermaid.so`. It can be called from any of the `*_ACCESS` tables. When called from the `PORT_ACCESS` table, it can be used to check limits based on the IP address of the connection which will be the most common way to implement MeterMaid as a replacement for the older `conn_throttle.so`. If called from other `*_ACCESS` tables, MeterMaid can also be used to establish limits on other data such as the envelope from or envelope to addresses as well as IP addresses.

This chapter only describes the throttle entry point in `check_metermaid.so`. For a complete list of

`check_metermaid.so` entry points, refer to "MeterMaid Reference" in *Messaging Server Administration Reference*. The `throttle` routine contacts MeterMaid providing two subsequent arguments separated by commas. The first is the name of the table against which the data will be checked, and the second is the data to be checked.

If the result from the probe is that the particular data being checked has exceeded its quota in that table, `check_metermaid.so` returns "success" so that the mapping engine will continue processing this entry. The remainder of the entry would then be used to handle this connection that has exceeded its quota.

```
PORT_ACCESS

*|*|*|*|* $C$|INTERNAL_IP;$3|$Y$E
*|*|*|*|*
$C$:A$[/opt/sun/comms/messaging64/lib/check_metermaid.so,throttle,tablename
\
Connection$ declined$ at$ this$ time$E
* $YEXTERNAL
```

Note the `$:A` flag test in the mapping table entry before the call to `check_metermaid.so.` This is to ensure that we only do the MeterMaid probe when `PORT_ACCESS` is being checked by the dispatcher as it will set the `A` flag for its probe.

Options for MeterMaid

MeterMaid's configuration is maintained by setting options using the `msconfig` command in Unified Configuration or using the `configutil` command in legacy configuration. The following table describes some of the settings currently supported by MeterMaid.

Options for MeterMaid

Unified Configuration Option	Legacy Configuration Option	Description
`metermaid.enable`	`local.metermaid.enable`	This setting must be set to `1` on the system that will run the MeterMaid daemon so that the Watcher will start and control MeterMaid.
`metermaid.logfile.*`	`logfile.metermaid.*`	These settings are the same as those used by imap, pop, and other services. By default MeterMaid writes its log file into *msg-svr-base* `/data/log/metermaid`.
`metermaid.listenaddr`	`metermaid.config.listenaddr`	The address to which MeterMaid should bind. On most systems, the default would not need to be changed, but for multi-homed or HA systems, specifying the appropriate address here is recommended. Default: (`INADDR_ANY`)
`metermaid.maxthreads`	`metermaid.config.maxthreads`	The MeterMaid server is multithreaded and maintains a pool of threads onto which its tasks are scheduled. This value sets the maximum number of threads that will be used by MeterMaid. On systems with more than 4 CPUs, increasing this value may increase overall throughput. Default: 20
`metermaid.port`	`metermaid.config.port`	This is the port to which MeterMaid listens for connections and to which MeterMaid clients will connect. Default: 63837
`metermaid.secret`	`metermaid.config.secret`	In order to authenticate incoming connections, MeterMaid uses a shared secret that the clients send once they connect to MeterMaid. No default. Value must be supplied.
`metermaid.sslusessl`	`service.metermaid.sslusessl`	Requires the use of SSL for all incoming connections to the MeterMaid server. Default: 0

The following table describes the options used by the `check_metermaid` client:

Options Used by the `check_metermaid` Client

Unified Configuration Option	Legacy Configuration Option	Description
metermaid_client. connectfrequency	metermaid.mtaclient. connectfrequency	Attempt a connection every connectfrequency seconds. When the client needs to connect to MeterMaid, it uses this as an internal throttle to prevent constant connection attempts when MeterMaid isn't available. During the time that the client is unable to communicate with MeterMaid, it will return a "fail" status to the MTA mapping engine indicating that MeterMaid has not blocked this connection. For example, if check_metermaid.so attempts to connect to MeterMaid, but it fails for some reason, during the next N seconds as specified by metermaid_client.connectfrequency (or metermaid.mtaclient.connectfrequency in legacy configuration), no additional attempts will be attempted. It prevents check_metermaid.so from trying to connect to MeterMaid too frequently if it is not working. Default: 15
metermaid_client. connecttimeout	metermaid.mtaclient. connectwait	When the client is waiting for a connection to MeterMaid (either an initial connection or to reuse another already established connection), it will wait for connecttimeout seconds before returning a fail status and allowing this connection to continue. Default: 5
metermaid_client. debug	metermaid.mtaclient. debug	If this option is enabled, debugging information from the client will be printed into either the server or thread-specific log file for the SMTP server. Default: no
metermaid_client. max_conns	metermaid.mtaclient. maxconns	In order to support multithreaded servers, the client can maintain a pool of connections to MeterMaid. By doing this, there can be increased concurrency during communications. However, due to internal locking done by MeterMaid, access to a particular table is limited to one request at a time, so multiple connections from a single process may provide limited benefit. Default: 3
metermaid_client. timeout	metermaid.mtaclient. readwait	When communicating with MeterMaid, the client will wait timeout seconds before returning a fail status and allowing this connection to continue. Default: 10
metermaid_client. server_host	metermaid.config. serverhost	This is the host name or IP address to which the clients will connect. It may be the same as metermaid.listenaddr but will most likely have a particular value to direct clients to one system in particular in the Messaging Server environment. No default. Value must be supplied.
metermaid_client. sslusessl	metermaid.mtaclient. sslusessl	Enables SSL communication with an SSL-enabled MeterMaid server. Default: 0

Lastly, the throttling tables are also defined in Unified and legacy configurations. The following table describes the options defining the throttling tables. The * in each configuration parameter is the name of

the particular table being defined. For example, for a table called internal, the first parameter would be called `metermaid.local_table:internal.data_type` in Unified Configuration or `metermaid.table.internal.data_type` in legacy configuration.

Options for Defining Tables

Unified Configuration Option	Legacy Configuration Option	Valid for Table Types	Description
`metermaid.local_table:*.data_type`	`metermaid.table.*.data_type`	throttle simple greylisting	MeterMaid can support two kinds of data in its tables, string and ipv4. string data is limited to 255 bytes per entry and can be compared using case-sensitive or case-insensitive functions (see `metermaid.local_table:*.options` below). Default: string
`metermaid.local_table:*.max_entries`	`metermaid.table.*.max_entries`	throttle simple greylisting	When MeterMaid initializes each table, it pre-allocates this many entries. MeterMaid automatically recycles old entries, even if they haven't yet expired. When a new connection is received, MeterMaid will reuse the least recently accessed entry. A site should specify a value high enough to cache the connections received during `quota_time`. Default: 1000

`metermaid.local_table:*.options`	`metermaid.table.*.options`	throttle	This option is a comma-separated list of keywords that defines behaviour or characteristics for the table. Valid keywords are: • `nocase` - When working with the data, all comparisons are done using a case-insensitive comparison function. (This option is valid only for string data). • `penalize` - After *quota_time* seconds, throttle will normally reset the connection count to 0, but if the penalize option is enabled, throttle will decrement the connection count by quota (but not less than 0) so that additional connection attempts will penalize future `quota_time` periods. For example, if quota were 5 with a `quota_time` of 60, and the system received 12 connection attempts during the first minute, the first 5 connections would be accepted and the remaining 7 would be declined. After 60 seconds has passed, the number of connections counted against the particular address would be reduced to 7, still keeping it above quota and declining connection attempts. Assuming no additional connection attempts were made, after another 60 second period, the number of connections would be further reduced down to 2, and MeterMaid would permit connection attempts again.
`metermaid.local_table:*.quota`	`metermaid.table.*.quota`	throttle	When a connection is received, it is counted against quota. If the number of connections received in `quota_time` seconds exceeds this value, MeterMaid will decline the connection. (The actual effect on the incoming connection is controlled by the mapping table and could result in additional scrutiny, a delay, or denying the connection.) Default: 100
`metermaid.local_table:*.quota_time`	`metermaid.table.*.quota_time`	throttle	This specifies the number of seconds during which connections will be counted against `quota`. After this many seconds, the number of connections counted against the incoming address will be reduced depending on the `type` of this table. Default: 60

`metermaid.local_table:*.storage`	`metermaid.table.*.storage`	throttle simple greylisting	MeterMaid can use two different storage methods, `hash` and `splay`. The default hash table method is recommended, but under some circumstances a splay tree may provide faster lookups. Default: hash
`metermaid.local_table:*.table_type`	`metermaid.table.*.type`	NA	MeterMaid supports three table types: • `throttle` (default) - This type of table keeps track of the data, typically IP addresses, and will throttle the incoming connections to `quota` connections during a period of `quota_time` seconds. • `simple` - This type of table may be used to store arbitrary data referenced by a key. • `greylisting` - This type of table may be used to provide an anti-spam/anti-virus technique. More information about setting up this type of table can be found at in Implementing Greylisting by Using MeterMaid.
`metermaid.local_table:*.block_time`	`metermaid.table.*.block_time`	greylisting	Specifies the ISO 8601 duration for how long we temporarily reject each delivery attempt based on sender and recipient information. Default: pt5m (5 minutes)
`metermaid.local_table:*.resubmit_time`	`metermaid.table.*.resubmit_time`	greylisting	Specifies the ISO 8601 duration during which, but after `block_time`, we must receive a subsequent delivery attempt based on the same sender and recipient information previously blocked. This sender and recipient combination is now flagged as permitted. Default: pt4h (4 hours)
`metermaid.local_table:*.inactivity_time`	`metermaid.table.*.inactivity_time`	greylisting	Specifies the ISO 8601 duration for how long we will continue to accept messages based on the sender and recipient information previously permitted. This permission expires after `inactivity_time` from the last allowed delivery. Default: p7d (7 days)

Options Used to Enable Access Multiple MeterMaid Servers from `check_metermaid.so`

Unified Configuration Option	Legacy Configuration Option	Description
`metermaid_client.remote_table:`*table*`.server_nickname`	`metermaid.mtaclient.remote_table.`*table*`.server_nickname`	Specifies the nickname for a particular MeterMaid server that is responsible for the referenced *table*. The nickname is a short keyword, consisting only of letters, numbers, and underscores, that will be used in the `remote_server` option names. No default. Value must be supplied.
`metermaid_client.remote_server:`*nickname*`.max_conns`	`metermaid.mtaclient.remote_server.`*nickname*`.max_conns`	Specifies the maximum number of concurrent connections to the MeterMaid server referenced by *nickname*. Default: 3
`metermaid_client.remote_server:`*nickname*`.server_host`	`metermaid.mtaclient.remote_server.`*nickname*`.server_host`	Specifies the host name of the MeterMaid server referenced by *nickname*. Defaults to the value of `metermaid_client.server_host`.
`metermaid_client.remote_server:`*nickname*`.server_port`	`metermaid.mtaclient.remote_server.`*nickname*`.server_port`	Specifies the port number of the MeterMaid server referenced by *nickname*. Default: 63837
`metermaid_client.remote_server:`*nickname*`.sslusessl`	`metermaid.mtaclient.remote_server.`*nickname*`.sslusessl`	Specifies whether or not to use SSL for the connection to the MeterMaid server referenced by *nickname*. Defaults to the value of `metermaid_client.sslusessl`.

Limit Excessive IP Address Connections Using Metermaid – Example

This example uses MeterMaid to throttle IP addresses at 10 connections/minute. For reference, the equivalent `conn_throttle.so` setup in the mappings table would be as follows:

```
PORT_ACCESS

*|*|*|*|* $C$|INTERNAL_IP;$3|$Y$E
*|*|*|*|*
$C$[/opt/sun/comms/messaging/lib/conn_throttle.so,throttle,$3,10]\
$N421$ Connection$ declined$ at$ this$ time$E
* $YEXTERNAL
```

This `PORT_ACCESS` mapping table implements `conn_throttle.so` to restrict connections to a rate of no more than 10 connections per minute for non-INTERNAL connections.

One fundamental difference between the two technologies is that instead of configuring details such as the rate-limit for throttling directly into the mapping table, MeterMaid uses `msconfig` for these settings, as the following example shows.

On systems running the MeterMaid server:

1. Enable MeterMaid by running the following command:
   `msconfig set metermaid.enable 1`
   or in legacy configuration:
   `configutil -o local.metermaid.enable -v TRUE`
2. Set an authentication password used to verify communications between the client and MeterMaid server:

`msconfig set metermaid.secret` *password*
or in legacy configuration
`configutil -o metermaid.config.secret -v` *password*

3. Define a throttling table
   MeterMaid's throttling behavior is determined by the use of named throttling tables that define operating characteristics. To define a table that throttles at a rate of 10 connections per minute, set the following parameters in `msconfig`:
   `set metermaid.local_table:ext_throttle.data_type` ipv4
   `set metermaid.local_table:ext_throttle.quota` 10
   or the following for legacy configuration using `configutil`:
   `configutil -o metermaid.table.ext_throttle.data_type -v ipv4`
   `configutil -o metermaid.table.ext_throttle.quota -v 10`
   ext_throttle is the name of the throttling table. `ipv4` is the data type Internet Protocol version 4 address representation. 10 is the quota (connection limit).
4. On the MeterMaid system, start MeterMaid.
   `start-msg metermaid`
5. On systems where the MTA will use MeterMaid to do throttling, specify the MeterMaid host and password.
   These are required:
   `msconfig set metermaid.secret` *MeterMaid_Password*
   `msconfig set metermaid_client.server_host` *name_or_ipaddress_of_MetermaidHost*
   or the following for legacy onfiguration:
   `configutil -o metermaid.config.secret -v` *MeterMaid_Password*
   `configutil -o metermaid.config.serverhost -v` *name_or_ipaddress_of_MetermaidHost*
6. Set up the MeterMaid `PORT_ACCESS` table.
   This table is similar to the equivalent `conn_throttle.so` setup:

```
PORT_ACCESS

*|*|*|*|* $C$|INTERNAL_IP;$3|$Y$E
*|*|*|*|*
$C$:A$[/opt/sun/comms/messaging/lib/check_metermaid.so,throttle,\
ext_throttle,$3]$N421$ Connection$ declined$ at$ this$ time$E
* $YEXTERNAL
```

   The first line checks to see if the IP address attempting a connection is internal. If it is, it allows the connection. The second line runs the IP address through MeterMaid and if it has connected too frequently, it declines the connection. The third line allows any other connections through, but flagged as `EXTERNAL`.
   This call to `check_metermaid.so` is very similar to the callout to `conn_throttle.so`. The function in `check_metermaid.so` is the same. `throttle` and its arguments are simply the table name as configured using `metermaid.local_table:`*tablename* and the IP address to check ($3). Like `conn_throttle.so`, this function returns *success* when the limit (as specified in `metermaid.local_table:ext_throttle.quota`) has been reached. This allows the remainder of the mapping entry line is processed, which sends a message (421 SMTP code, transient negative completion, *Connection not accepted at this time*) to the remote SMTP client, and tells the Dispatcher to close the connection.
   `$:A` ensures that this line will only be processed when being called from the Dispatcher. Without this, the call to `check_metermaid.so` would also happen in the context of the `tcp_smtp_server` processes which also probes the `PORT_ACCESS` mapping table. This would cause MeterMaid to count each incoming connection twice.

This is the basic configuration to set up MeterMaid as a `conn_throttle.so` replacement. See Mapping Operations and PORT_ACCESS Mapping Table for information on these topics.

Chapter 63. Sun Gathering Debug Data for Sun Java System Messaging Server

Sun Gathering Debug Data for Sun Java System Messaging Server

This technical note describes how gather data to facilitate debugging problems with a Sun Java System Messaging Server system. By collecting this data before you open a Service Request, you can reduce substantially the time needed to analyze and resolve the problem. For more information on how this document and associated scripts can help you in better dealing with Messaging Server problems, see Sun Gathering Debug Data.

This document is intended for anyone who needs to open a Service Request about Messaging Server with My Oracle Support.

This technical note contains the following sections:

About This Technical Note

This document covers the following versions of Sun Java System Messaging Server on the Solaris Operating System, HP-UX, Red Hat Linux, and Microsoft Windows platforms:

- Sun Java System Messaging Server 7
- Sun Java System Messaging Server 6.3
- Sun Java System Messaging Server 6 2005Q4
- Sun Java System Messaging Server 6 2005Q1
- Sun Java System Messaging Server 6 2004Q2
- Sun Java System Messaging Server 6 2003Q4
- Sun ONE Messaging Server 6
- iPlanet Messaging Server 5

You can use this document in all types of environments, including test, pre-production, and production. Verbose debugging is not used (to reduce performance impact), except when it is deemed necessary. At the same time, it is possible that the problem could disappear when you configure logging for debug mode. However, this is the minimum to understand the problem. In the majority of cases, the debug data described in this document is sufficient to analyze the problem.

This document does not provide workarounds nor techniques or tools to analyze debug data. It provides some troubleshooting, but you should not use this guide as an approach to troubleshooting Messaging Sever problems.

If your problem does not conveniently fit into any of the specific categories, supply the general

information described in What Messaging Server Debug Data Should You Collect? and clearly explain your problem.

If the information you initially provide is not sufficient to find the root cause of the problem, Oracle will ask for more details, as needed.

Prerequisites for Collecting Messaging Server Debug Data

The prerequisites for collecting debug data for Messaging Server are as follows:

- Make sure you have `superuser` privileges.
- For the Solaris OS and Red Hat Linux platforms, obtain the `dbhang` and `pkgapp` scripts from My Oracle Support. See Running the Messaging Server Debugging Scripts for more information.
- On the Windows platform, download the free Debugging Tools for Windows to help in analyzing process hang problems. The debugger Dr. Watson is not useful for process hang problems because it cannot generate a crash dump on a running process. Download the free Debugging Tools from http://www.microsoft.com/whdc/devtools/debugging/default.mspx.

Install the last version of Debugging Tools and the OS Symbols for your version of Windows. Also, you must add the environment variable `NT_SYMBOL_PATH`.

Use the command `drwtsn32 -i` to select Dr. Watson as the default debugger. Use the command `drwtsn32`, check all options, and choose the path for crash dumps.

Variables Used in This Technical Note

The following describes the variables used in the procedures in this document. Gather the values of the variables if you don't already know them before you try to do the procedures.

- *channel*: The channel name where the messages are queued.
- *identifier*: The Messaging Server instance name used during installation. The installation program automatically added the prefix `msg-` to the name you specified. For example, if you named the identifier `tango`, the installation program created `msg-tango`. This only applies to iPlanet Messaging Server 5.
- *messaging-pid*: Process ID of a Messaging Server daemon.
- *messaging-service-port*: Port number used by a Messaging Service (IMAP, POP, HTTP, and so forth).
- *msg-instance*: The directory on the Messaging Server machine dedicated to holding configuration, maintenance, and information files for a specific instance. This directory is located under *server-root*. This only applies to iPlanet Messaging Server 5. In Sun Java System Messaging Server 6, it is the `config` directory.
- *server-root*: The directory on the Messaging Server machine dedicated to holding the server program, configuration, maintenance, and information files. The default location for the Solaris OS version of Sun Java System Messaging Server 6 is `/opt/SUNWmsgsr/`. See To Obtain the server-root Variable for more information on determining the value of *server-root*.
- *windbg-root*: The directory on the Windows Messaging Server machine dedicated to holding the Win Debugger program, and configuration, maintenance, and information files.

To Obtain the server-root Variable

1. Use the following to obtain the value of the *server-root* variable.
   - Messaging Server (64bit) 7:

     Solaris OS

     ```
     pkgparam SUNWmessaging-server-64 BASEDIR
     ```

- Messaging Server (32bit) 7:

  Solaris OS

  ```
  pkgparam SUNWmessaging-server BASEDIR
  ```

  Red Hat Linux

  ```
  rpm -q --qf '%{INSTALLPREFIX}' sun-messsaging-server
  ```

- Sun Java System Messaging Server 64bit (Messaging Server 6):

  Solaris OS

  ```
  pkgparam SUNWmessaging-server-64 BASEDIR
  ```

- Sun Java System Messaging Server 32bit (Messaging Server 6):

  Solaris OS

  ```
  pkgparam SUNWmsgco BASEDIR
  ```

  Red Hat Linux

  ```
  rpm -q --qf '%{INSTALLPREFIX}' sun-messsaging-server
  ```

- iPlanet Messaging Server (Messaging Server 5):

  Solaris OS

  Look in the `/etc/msgregistry.inf` file.

  Windows

  Look in the `C:\windows\system32\etc\msgregistry.inf` file.

Overview of Collecting Debug Data for Messaging Server

Collecting debug data for a Messaging Server problem involves these basic operations:

1. Collecting basic problem and system information.
2. Collecting specific problem information (installation problem, process hang, process crash, and so on).
3. Creating a `tar.gz` file of all the information and uploading it for My Oracle Support.
4. Creating a Service Request with My Oracle Support.

Creating a Service Request with My Oracle Support

When you create a Service Request with My Oracle Support, provide the following information:

- Version info - the output from `imsimta version`
- A clear problem description:
- we should expand this to include some ATS basics
- Details of the state of the system, both before and after the problem started
- Impact on end users
- All recent software and hardware changes
- Any actions already attempted

- Whether the problem is reproducible; when reproducible, provide the detailed test case
- Whether a pre-production or test environment is available
- Name and location of the archive file containing the debug data

Once the Service Request is created, upload any additional files to https://supportfiles.sun.com/upload with the SR Number appended to the front of the file name.

Upload your debug data archive file to https://supportfiles.sun.com/upload.

For more information on how to upload files to this site, see the supportfiles FAQ.

> **Note**
> When opening a Service Request by phone with My Oracle Support, provide a summary of the problem, then give the details in a text file named `Description.txt`. Be sure to include `Description.txt` in the archive along with the rest of your debug data.

What Messaging Server Debug Data Should You Collect?

This section describes the kinds of debug data that you need to provide based on the kind of problem you are experiencing:

- To Collect Required Debug Data for Any Messaging Server Problem
- To Collect Debug Data on Messaging Server Installation Problems
- To Collect Debug Data on a Hung or Unresponsive Messaging Server Process
- To Collect Debug Data on a Messaging Server Crashed Process
- To Collect Debug Data on a Messaging Server Routing Problem
- To Collect Debug Data on a Messaging Server MTA Queue Problem
- To Collect Debug Data on a Messaging Server Webmail Problem

To Collect Required Debug Data for Any Messaging Server Problem

All problems described in this technical note need basic information collected about when the problem occurred and about the system having the problem. Use this task to collect that basic information.

1. Note the day(s) and time(s) the problem occurred.
2. Provide a graphical representation of your deployment. Include all hosts and IP addresses, host names, operating system versions, role they perform, and other important systems such as load balancers, firewalls, and so forth.
3. Note the operating system.

   Solaris OS

   ```
   uname -a
   ```

   HP-UX

   ```
   uname -r
   ```

   Red Hat Linux

   ```
   more /etc/redhat-release
   ```

   Windows

   ```
   C:\Program Files\Common files\Microsoft Shared\MSInfo\msinfo32.exe
   /report C:\report.txt
   ```

4. Note the patch level.

   Solaris OS

   `patchadd -p`

   HP-UX

   `swlist`

   Red Hat Linux

   `rpm -qa`

   Windows

   Already provided in the `C:\report.txt` file above.

5. Note the version of Messaging Server.

   Be sure to send the entire screen output of the `imsimta version` command.

   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     `cd` *server-root*`/sbin`
     `./imsimta version`

     Windows

     `cd` *server-root*`\sbin`
     `imsimta.exe version`

   - iPlanet Messaging Server (Messaging Server 5):

     UNIX and Red Hat Linux

     `cd` *server-root*`/msg-`*identifier*`./imsimta version`

     Windows

     `cd` *server-root*`\msg-`*identifier*`imsimta.exe version`

6. Create a tar file of the Messaging Server configuration directory.

   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     `cd` *server-root*`/sbin`
     `./configutil`
     Create a tar file of the *server-root*`/config` directory.

     Windows

     `cd` *server-root*`\config`
     `configutil.exe`
     Create a tar file of the *server-root*`\config` directory.

   - iPlanet Messaging Server (Messaging Server 5):

UNIX and Red Hat Linux

`cd` *server-root*`/msg-`*identifier*`./configutil`
Create a tar file of the *server-root*`/msg-`*identifier*`/imta/config` directory.

Windows

`cd` *server-root*`\msg-`*identifier*`configutil.exe`
Create a tar file of the *server-root*`\msg-`*identifier*`\imta\config` directory.

> **Note**
> If possible, provide just the relevant extracts of log files for the same time period that show the problem, with sufficient context to see what else was happening during the error occurrence and shortly before. Thus for relatively short log files (for example, MTA channel debug log files), send the entire log file, whereas for long-running hence large log files, an extract might be more appropriate, though be sure to include all the material from the time of the error as well as at least some lead-in logging from before the error apparently occurred.
>
> However, when questions arise about message structure or content, or about notification or bounced messages, then send an entire sample message, including all the outermost header (not just an excerpt of the message).

To Collect Debug Data on Messaging Server Installation Problems

Follow these steps if you are unable to complete the installation or if you get a "failed" installation status for Messaging Server.

1. Consult the following troubleshooting information:
   - Messaging Server 6 2005Q4: Chapter 9, Troubleshooting, in *Sun Java Enterprise System 2005Q4 Installation Guide for UNIX*
   - Messaging Server 6 2005Q1: Chapter 13, Troubleshooting, in *Sun Java Enterprise System 2005Q1 Installation Guide*
   - Messaging Server 6 2004Q2: Chapter 11, Troubleshooting, in *Sun Java Enterprise System 2004Q2 Installation Guide*
   - Messaging Server 6 2003Q4: Chapter 9, Troubleshooting Installation Problems, in *Sun Java Enterprise System 2003Q4 Installation Guide*
     If the problem persists after using this troubleshooting information, then continue with this procedure to collect the necessary data for My Oracle Support.
2. Collect the general system information as explained in To Collect Required Debug Data for Any Messaging Server Problem.
3. Specify if this is a first-time installation or a Hot Fix installation on a pre-existing Messaging Server instance.
4. Get the installation logs.
   - Sun Java System Messaging Server (Messaging Server 6):

     Messaging Server 6 log files mostly reside in the *server-root*`/log` directory. However, the initial configuration log files reside in the *server-root*`/install` directory, which also contains information on the initial configuration.

Solaris OS

`/var/sadm/install/logs`
The log file names start with `Java_Enterprise_System*install.B`datetime_, where *date* and _time _ correspond to the failing installing (for example, `B12161532`).

HP-UX and Red Hat Linux

`/var/opt/sun/install/logs`
The log file names start with `Java_Enterprise_System*install.B`datetime_, where *date* and _time _ correspond to the failing installing (for example, `B12161532`).

Windows

`C:\DocumentsandSettings`*current-user*`\LocalSettings\Temp`
The log file names start with `MSI*.log` (usually a text file). The asterisk (`*`) represents a random number in the `Temp` directory for each MSI based setup.

- iPlanet Messaging Server (Messaging Server 5): Rerun the installation with the following command and save the resultant file.

  Solaris OS

  ```
  truss -ealf -rall -wall -vall -o /tmp/install-messaging.truss
  ./setup
  ```

  HP-UX

  ```
  tusc -v -feaIT -rall -wall -o /tmp/install-messaging.tusc ./setup
  ```

  Red Hat Linux

  ```
  strace -fv -o /tmp/install-messaging.strace ./setup
  ```

  Windows

  Use Debug View:
  http://www.sysinternals.com/Utilities/DebugView.html

To Collect Debug Data on a Hung or Unresponsive Messaging Server Process

A process hang is defined as one of the Messaging Server processes not responding to requests anymore while the process is still running locally. Messaging Server's seven specific processes are:

- `imapd`---Provides access to IMAP services.
- `popd`---Provides access to POP services.
- `mshttpd`---Provides access to Webmail services.
- `dispatcher`---Permits multiple multithreaded server processes to share responsibility for SMTP connection services.
- `enpd`---Collects and dispatches events that occur to properties of resources (inboxes or calendars).
- `job_controller`---Ensures delivery of messages.
- `stored`---Performs a variety of important tasks such as deadlock and transaction operations of the message database, enforcing aging policies, and expunging and erasing messages stored on disk.

Additionally, Messaging Server uses these three processes:

- `tcp_smtp_server`---Handles SMTP sessions.
- `tcp_lmtp_server`---Handles LMTP sessions.
- `tcp_lmtpn_server`---Handles LMTP native sessions.

The system can be running more than one of each of these processes (especially `tcp_smtp_server`).

Other processes from the `job_controller.cnf` configuration file might be running (`autoreply`, `ims_master`, `l_master`, `conversion`, `reprocess`, and so on), as well as `reconstruct` and `imexpire` processes.

Before You Begin
Make sure that you collect all the data over the same time frame in which the problem occurs. See Configuring Solaris OS to Generate Core Files if a core file is not generated.

Collect the following information for process hang problems. Run the commands in order when the problem occurs. Be sure to specify the time when the process hang happened and affected processes, if possible.

1. Collect the general system information as explained in To Collect Required Debug Data for Any Messaging Server Problem.
2. Run the `netstat` command and save the output.

   UNIX and Red Hat Linux

   `netstat -an | grep` *messaging-service-port*

   Windows

   `netstat -an`

3. Run the following commands and save the output.

   Solaris OS

   ```
   ps -ef | grep server-rootvmstat 5 5
   iostat -x
   top
   uptime
   ```

   HP-UX

   ```
   ps -aux | grep server-rootvmstat 5 5
   iostat -x
   top
   sar
   ```

   Red Hat Linux

   ```
   ps -aux | grep server-rootvmstat 5 5
   top
   uptime
   sar
   ```

   Windows

   Obtain the `MESSAGING` process PID: `C:`*windbg-root*`>tlist.exe`
   Obtain process details of the `MESSAGING` running process PID:
   `C:`*windbg-root*`>tlist.exe` *messaging-pid*

4. Get the swap information.

   Solaris OS

   ```
   swap -l
   ```

   HP-UX

   ```
   swapinfo
   ```

   Red Hat Linux

   ```
   free
   ```

   Windows

   Already provided in `C:\report.txt` as described in To Collect Required Debug Data for Any Messaging Server Problem.

5. Get the system logs.

   Solaris OS and Red Hat Linux

   ```
   /var/adm/messages
   /var/log/syslog
   ```

   HP-UX

   ```
   /var/adm/syslog/syslog.log
   ```

   Windows

   Event log files:
   Start->Settings-> Control Panel -~~> Event Viewer~~> Select Log
   Then click Action-> Save log file as

   > **Note**
   > For UNIX systems, depending on your site's configuration of the `SNDOPR_PRIORITY option.dat` option and your syslog configuration (`syslog.conf`), the MTA might be sending automatically generated syslog notices to a non-default location. Also, the `LOG_SNDOPR option.dat` option controls whether additional potential syslog notices are generated by the MTA message logging facility.

6. For UNIX and Red Hat Linux systems, get the contents of the `/etc/nsswitch.conf` and `/etc/resolv.conf` files.
7. Get the most current log files for the hanging process, if known. Otherwise, get the log files for all processes.

   You should also set `LOG_SNDOPR` in the `option.dat` file and check for syslog notices if some expected MTA log file doesn't seem to have been generated. When you set the `LOG_SNDOPR` option, then the MTA generates a syslog notice if it cannot write to its regular log files.

   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     server-root`/log/*`

Windows

server-root\data\log\default\default *server-root*\data\log\http\http
server-root\data\log\imap\imap *server-root*\data\log\pop\pop
server-root\data\log\imta*

- iPlanet Messaging Server (Messaging Server 5):

  UNIX and Red Hat Linux

  server-root/msg-*identifier*/log/default/default *server-root*/msg-*identifier*/log/http/http *server-root*/msg-*identifier*/log/imap/imap *server-root*/msg-*identifier*/log/pop/pop
  server-root/msg-*identifier*/log/imta/*

  Windows

  server-root\msg-*identifier*\log\default\default
  server-root\msg-*identifier*\log\http\http
  server-root\msg-*identifier*\log\imap\imap *server-root*\msg-*identifier*\log\pop\pop
  server-root\msg-*identifier*\log\imta*

8. (Solaris OS only) If you are able to isolate the hanging process, get the following debug data for that process. Otherwise, get the following data for each of the Messaging Server processes.

   Using the PID obtained in Step 3, get a series of five of the following commands (one every 10 seconds):
   pstack *messaging--pid*
   pmap -x *messaging--pid*
9. Look for any core file that could have been dumped by one of the Messaging Server processes. If you find one, see To Collect Debug Data on a Messaging Server Crashed Process.
10. If any of the ims_master, imapd, mshttpd, popd, mboxutil, or reconstruct processes is hung, as the mailsrv user, get the outputs of the db_stat command as follows.
    - UNIX and Red Hat Linux:
      cd *msg--instance*/data/store/mboxlist
      server-{}~~*root*/lib/db_stat -C o -h *msginstance*/data/store/mboxlist/ -~~N
      server-{}~~*root*/lib/db_stat -~~t
      If you have set the dbtmpdir configutil variable, use that location instead of *msg-instance*/store/mboxlist/.

      > **Note**
      > For Solaris OS systems only, the script dbhang captures all the following debug data for you. Edit the top of the script to match your system's configuration. Specifically, edit the SRVROOT, INST, MAILUSER, and DBTMPDIR parameters. Then run the script and collect the data. See Running the Messaging Server Debugging Scripts for more information.

    - Windows:
      cd *msg--instance*\data\store\mboxlist
      server-{}~~*root*\lib\db_stat -C o -h *msginstance*\data\store\mboxlist\ -~~N
      server-{}~~*root*\lib\db_stat -~~t
11. Get the output of the following command.

    Solaris OS

    truss -ealf -rall -wall -vall -o /tmp/truss.out -p *messaging-pid*

HP-UX

```
tusc -v -fealT -rall -wall -o /tmp/tusc.out -p messaging-pid
```

Red Hat Linux

```
strace -fv -o /tmp/strace.out -p messaging-pid
```

Windows

Use DebugView:
http://www.sysinternals.com/Utilities/DebugView.html

> **Note**
> Wait one minute after launching the appropriate command (`truss`, `strace`, `tusc`, or DebugView) then stop it by pressing **`Control-C`** in the terminal where you launched the command.

12. Get core files and the output of the following commands.

    In a process hang situation, it is helpful to compare several core files to review the state of the threads over time. To not overwrite a core file, copy that core file to a new name, wait approximately one minute then rerun the following commands. Do this three times to obtain three core files.

> **Note**
> For HP-UX, you need the following two patches to use the `gcore` command: PHKL_31876 and PHCO_32173. If you cannot install these patch, use the HP-UX `/opt/langtools/bin/gdb` command from version 3.2 and later, or the `dumpcore` command.

Solaris OS

```
cd server-root/bin/slapd/server
gcore -o /tmp/messaging_process-core messaging-pidpstack
/tmp/messaging_process-core
```

HP-UX

```
# gcore -{}p messaging-pid(gdb) attach messaging--pid
Attaching to process messaging--pid
No executable file name was specified
(gdb) dumpcore
Dumping core to the core file core.messaging--pid
(gdb) quit
The program is running. Quit anyway (and detach it)? (y or n) y
Detaching from program: , process messaging--pid
```

Red Hat Linux

```
# gdb
(gdb) attach messaging--pid
Attaching to process messaging--pid
No executable file name was specified
(gdb) gcore
Saved corefile core.messaging--pid
(gdb)backtrace (gdb)quit
```

Windows

Get the MESSAGING process PID:

`C:windbg-root>tlist.exe`

Generate a crash dump on the MESSAGING running process PID:

`C:windbg-root>adplus.vbs -hang -p messaging-pid -o C:\crashdump_dir`

> **Note**
> For Windows, provide the complete generated folder under `C:\crashdump_dir`.

13. (Solaris OS only) Archive the result of the script `pkgapp` (one core file is sufficient). `./pkgapp.ksh pid-{}~~of~~` application *corefile* My Oracle Support must have the output from the `pkgapp` script to properly analyze the core file(s).

> **Note**
> Make sure the appropriate limitations are set by using the `ulimit` command, and that the user is not `nobody`. Also check the `coreadm` command for additional control. See Configuring Solaris OS to Generate Core Files if a core file is not generated.

14. When you have collected all debug data, perform the following steps to restore the service. Messaging Server processes usually hang because of an orphan lock left in one of the databases. Stopping the server (especially the `stored` process), and cleaning the temporary shared db files helps to resolve the problem.
    a. Stop Messaging Server.

       `cd server-root/sbin`
       `./stop-msg`
    b. Make sure that all Messaging Server processes stopped.

       Wait one minute, then kill any remaining processes, except `tcp_smtp_server` processes (which do not use databases).
    c. Restart Messaging Server.

       `./start-msg`
    d. After restarting the services, check the logs for any unexpected behavior.

To Collect Debug Data on a Messaging Server Crashed Process

Use this task to collect data when a Messaging Server process has stopped (crashed) unexpectedly. Run all the commands on the actual machine where the core file(s) were generated.

1. Collect the general system information as explained in To Collect Required Debug Data for Any Messaging Server Problem.
2. Note whether you can you restart Messaging Server.
3. Get the output of the following commands.

   Solaris OS

   ```
   ps -ef | grep server-rootvmstat 5 5
   iostat -x
   top
   uptime
   ```

   HP-UX

   ```
   ps -aux | grep server-rootvmstat 5 5
   iostat -x
   top
   sar
   ```

   Red Hat Linux

   ```
   ps -aux | grep server-rootvmstat 5 5
   top
   uptime
   sar
   ```

   Windows

   Obtain the `MESSAGING` process PID: `C:`*windbg-root*`>tlist.exe`

   Obtain process details of the `MESSAGING` running process PID: `C:`*windbg-root*`>tlist.exe` *messaging-pid*

4. Get the swap information.

   Solaris OS

   ```
   swap -l
   ```

   HP-UX

   ```
   swapinfo
   ```

   Red Hat Linux

   ```
   free
   ```

   Windows

   Already provided in `C:\report.txt` as described in To Collect Required Debug Data for Any Messaging Server Problem.

5. Get the system logs.

   Solaris OS and Red Hat Linux

   ```
   /var/adm/messages
   /var/log/syslog
   ```

HP-UX

`/var/adm/syslog/syslog.log`

Windows

Event log files: Start->Settings-> Control Panel --~~> Event Viewer~~> Select Log Then click Action-> Save log file as

> **Note**
> For UNIX systems, depending on your site's configuration of the `SNDOPR_PRIORITY option.dat` option and your syslog configuration (`syslog.conf`), the MTA might be sending automatically generated syslog notices to a pre---determined location. Also, the `LOG_SNDOPR option.dat` option controls whether additional potential syslog notices are generated by the MTA message logging facility.

6. Get core files (called "Crash Dumps" by Windows).

   Solaris OS

   See Configuring Solaris OS to Generate Core Files if a core file was not generated.

   Red Hat Linux

   See http://kbase.redhat.com/faq/docs/DOC-5353.

   Windows

   Generate a crash dump during a crash of Messaging Server by using the following commands:

   Get the MESSAGING process PID : `C:`*windbg-root*`>tlist.exe` Generate a crash dump when the MESSAGING process crashes: `C:`*windbg-root*`>adplus.vbs -crash -FullOnFirst -p` *messaging-pid* `-o C:crashdump_dir`

   The `adplus.vbs` command watches *messaging-pid* until it crashes and will generate the `dmp` file. Provide the complete generated folder under `C:`*crashdump_dir*.

> **Note**
> If you didn't install the Debugging Tools for Windows, you can use the `drwtsn32.exe -i` command to select Dr. Watson as the default debugger. Use the `drwtsn32.exe` command, check all options, and choose the path for crash dumps. Then provide the dump and the `drwtsn32.log` files.

7. (Solaris OS only) For each core file, provide the output of the following commands.
   `file` *`corefile`*
   `pstack` *`corefile`*
   `pmap _corefile`
   `pflags` *`corefile`*
8. (Solaris OS only) Archive the result of the script `pkgapp` (one core file is sufficient).
   `./pkgapp.ksh` *`Pid`*`-{}`~~*`of`*~~ application *corefile*

Note
My Oracle Support must have the output from the `pkgapp` script to properly analyze the core file(s).

To Collect Debug Data on a Messaging Server Routing Problem

Use this task to collect data when Messaging Server is experiencing a routing problem.

A Messaging Server routing problem is defined as the inability of the system to correctly route a message. For example, a message might be ending up in the wrong Message Store, might be sent to the wrong channel, might be delivered to the wrong user, and so on.

1. Collect the general system information as explained in To Collect Required Debug Data for Any Messaging Server Problem.
2. Provide a detailed explanation of what do you want to obtain.
3. Get the output of the following command.
   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     ```
     cd server-root/sbin
     ./imsimta test -rewrite -debug=level=5 mailaddress
     ```

     Windows

     ```
     cd server-root\sbin
     imsimta.exe test -rewrite -debug=level=5 mailaddress
     ```

   - iPlanet Messaging Server (Messaging Server 5):

     UNIX and Red Hat Linux

     ```
     cd server-root/msg-identifier./imsimta test -rewrite -debug=level=5
     mailaddress
     ```

     Windows

     ```
     cd server-root\msg-identifierimsimta.exe test -rewrite -debug=level=5
     mailaddress
     ```

Note
When the problem is about Sieve filters, add the option -filter to the above command.

The -noimage qualifier to the `imsimta test -rewrite -debug` command enables you to test changes made to the configuration file prior to recompiling the new MTA configuration.

If possible, use the -source_channel=*source_channel* option to specify the incoming channel. This is sometimes necessary for testing the interactions with the mapping tables and the antirelay rules. By default, Internet mail arrives on the `tcp_local` channel whereas internal mail arrives on the `tcp_intranet` channel. If the connection is authenticated, use `tcp_auth`.

The command would then look like: `# ./imsimta test -rewrite -debug -source_channel=tcp_intranet` *email-address*

4. Get the LDIF entry of an impacted user.

   UNIX and Red Hat Linux

   dir-root`/shared/bin/ldapsearch -h` *hostname* `__ }}{{-p __`*port* `-D "cn=Directory Manager" -w` *password* `-b "`*basedn*`" "(objectclass=*)" uid=user > /tmp/user.ldif`

   Windows

   dir-root`\shared\bin\ldapsearch.exe -h` *hostname* `-p` *port* `-D "cn=Directory Manager" -w` *password* `__ }}{{-b "_basedn{_}" "(objectclass=*)" uid=user > C:\user.ldif`

   where:

   dir-root

   The directory on the Directory Server machine dedicated to holding the server program, and configuration, maintenance, and information files. The default location for UNIX and Red Hat Linux versions of Messaging Server is `/var/opt/mps/serverroot/`.

   hostname

   Name of the host running Directory Server. The default value is `localhost`. You can omit -h *hostname* if the Directory Server is running locally.

   port

   Port number on which Directory Server is listening. The default is 389. You can omit *port* if the Directory Server is running on port 389.

   basedn

   The base dn for the search. Use _ basedn_ as the starting point for the search.

5. Get the LDIF entry of the domain where the impacted user resides.

UNIX and Red Hat Linux

dir-root`/shared/bin/ldapsearch -h` *hostname* `-p` *port* `-D "cn=Directory Manager" -w` *password* `-s base -b "`*baseDN*`" "(objectclass=*)" > /tmp/domain.ldif`

Windows

dir-root`\shared\bin\ldapsearch.exe -h` *hostname* `-p` *port* `-D "cn=Directory Manager" -w` *password* `__ }}{{-s base -b "_baseDN{_}" "(objectclass=*)" > C:\domain.ldif`

where:

dir-root

The directory on the Directory Server machine dedicated to holding the server program, and configuration, maintenance, and information files. The default location for UNIX and Red Hat Linux versions of Messaging Server is `/var/opt/mps/serverroot/`.

hostname

Name of the host running Directory Server. You can omit -h *hostname* if the Directory Server is running locally.

port

Port number on which Directory Server is listening. The default is 389. You can omit *port* if the Directory Server is running on port 389.

To Collect Debug Data on a Messaging Server MTA Queue Problem

Use this task to collect data when Messaging Server is experiencing a queue problem, for example, when a specific queue is growing and messages are not being dequeued.

1. Collect the general system information as explained in To Collect Required Debug Data for Any Messaging Server Problem.
2. Is the "growing" queue full of `ZZ*.00` message files or full of `Z*.00` message files? How many `ZZ*.00` message files, and how many `Z*.00` message files are there? Or are there `.HELD` files?
    a. If the channel has lots of `ZZ*.00` message files and relatively few `Z*.00` message files (where "relatively" depends heavily on the specific channel and site usage), make sure that the Job Controller is running. For example:
    `ps --ef | grep job_controller`
    b. If a channel has lots of `Z*.00` message files and not very many `ZZ*.00` message files, then typically the MTA itself does not have any "problem," but rather than there is a problem with a separate destination host or a network problem.

    In this case, look at the delivery history of the `Z*.00` messages. You need the message files themselves, or better yet, the output of the `imsimta qm history` command. Examine the `imsimta qm history` output for old `mail.log*` records for those message files. They indicate what sort of SMTP or other error is occurring (and when) for these "old" message files.

    For more information on the `imsimta qm` command, see the following:
    - Messaging Server 6 2005Q4: "imsimta qm" in *Sun Java System Messaging Server 6 2005Q4 Administration Reference*
    - Messaging Server 6 2005Q1: "imsimta qm" in Chapter 2, "Message Transfer Agent Command-line Utilities," in *Sun Java System Messaging Server 6 2005Q1*

Administration Reference
- Messaging Server 6 2004Q2: "imsimta qm" in Chapter 2, "Message Transfer Agent Command-line Utilities," in *Sun Java System Messaging Server 6 2004Q2 Administration Guide*

3. Look at the "Messages are Not Dequeued" section in the *Messaging Server Administration Guide*.
   - Messaging Server 6 2005Q4: Chapter 22, Troubleshooting the MTA, in *Sun Java System Messaging Server 6 2005Q4 Administration Guide*
   - Messaging Server 6 2005Q1: Chapter 13, "Troubleshooting," in *Sun Java System Messaging Server 6 2005Q1 Administration Guide*
   - Messaging Server 6 2004Q2: Chapter 11, "Troubleshooting," in *Sun Java System Messaging Server 6 2004Q2 Administration Guide*
4. Run the following `imsimta` command to see if messages will now get delivered.

   If the messages do now get delivered, then whatever problem was preventing message delivery was probably transient (for example, a network or DNS problem) and is now resolved. Or else the "problem" is simply that you do not have the channel/Job Controller configured for enough simultaneous delivery jobs for that channel to keep up with the current load.
   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     `cd` *server-root*`/sbin`
     `./imsimta run` *channel*

     Windows

     `cd` *server-root*`\sbin`
     `imsimta.exe run` *channel*

   - iPlanet Messaging Server (Messaging Server 5):

     UNIX and Red Hat Linux

     `cd` *server-root*`/msg-`*identifier*`./imsimta run` *channel*

     > **Note**
     > The `imsimta run` command does not provide much helpful information for `Z*.00` message files.

5. Get the output of the following commands.
   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     `cd` *server-root*`/sbin`
     ```
     ./imsimta qm counters show
     ./imsimta qm summ
     ./imsimta qtop -database -domain_to
     ./imsimta qm messages channel
     ```

     Windows

     `cd` *server-root*`\sbin`
     ```
     imsimta.exe qm counters show
     imsimta.exe qm summ
     imsimta.exe qtop -database -domain_to
     imsimta.exe qm messages channel
     ```

- iPlanet Messaging Server (Messaging Server 5):

  UNIX and Red Hat Linux

  ```
  cd server-root/msg-identifier./imsimta qm counters show
  ./imsimta qtop -database -domain_to
  ```

  Windows

  ```
  cd server-root\msg-identifierimsimta.exe qm counters show
  imsimta.exe qm summ
  imsimta.exe qtop -database -domain_to
  ```

6. Get the current log files.
   - Sun Java System Messaging Server (Messaging Server 6):

     UNIX and Red Hat Linux

     server-root`/log/*`

     Windows

     server-root{{\data\log\imta
     *}}

   - iPlanet Messaging Server (Messaging Server 5):

     UNIX and Red Hat Linux

     server-root`/msg-`*identifier*`/log/imta/*`

     Windows

     server-root`\msg-`*identifier*{{\log\imta
     *}}

To Collect Debug Data on a Messaging Server Webmail Problem

Use this task to collect data for a Webmail problem. The most common problems are related to incorrect translation of fields when using a localized Messaging Server interface.

1. Collect the general system information as explained in To Collect Required Debug Data for Any Messaging Server Problem.
2. Take a snapshot of the problematic screen(s).
3. Note the step-by-step procedure to reproduce the problem with a test case.

> **Note**
> My Oracle Support does not support Webmail customizations. Contact your sales representative for those problems.

Configuring Solaris OS to Generate Core Files

Core files are generated when a process or application terminates abnormally. Core files are managed with the `coreadm` command. This section describes how to use the `coreadm` command to configure a system so that all process core files are placed in a single system directory. This means it is easier to track problems by examining the core files in a specific directory whenever a Solaris OS process or daemon terminates abnormally.

Before configuring your system for core files, make sure the file system chosen to hold the core files (`/var` in the following example) has sufficient space. Once you configure Solaris OS to generate core files, all processes that crash will write a core file to that directory.

To Configure Solaris OS to Generate Core Files

1. Run the following commands as `root`.

```
mkdir -p /var/cores
coreadm -g /var/cores/%f.%n.%p.%t.core \
    -e global -e global-setid -e log \
    -d process -d proc-setid
```

   The preceding command stores all core dumps in a central location with names identifying what process dumped core and when. These changes only impact processes started after you run the `coreadm` command. Use `coreadm -u` to apply the settings to all existing processes. For more information see the coreadm article on the FAQ site.

2. Set the size of the core dumps to unlimited.
   `# ulimit --c unlimited`
   `# ulimit --a`
   `coredump(blocks) unlimited`
   See the `ulimit` man page for further information.
3. Verify core file creation.

```
# cd /var/cores
# sleep 100000 &
[1]_PID_
# kill -8 _PID_
#
_PID_ Arithmetic Exception - core dumped
# ls
sleep.s4u-420rb-zone3-bur02.29352.1240596673.core
# tail /var/adm/messages
Apr 24 14:11:13 s4u-420rb-zone3-bur02 genunix: [ID 603404 kern.notice]
NOTICE: core_log: sleep[29352] core dumped:
/var/cores/sleep.s4u-420rb-zone3-bur02.29352.1240596673.core
```

Running the Messaging Server Debugging Scripts

This section describes how to run the `dbhang` and `pkgapp` scripts.

Download the Current Scripts from My Oracle Support

The `dbhang` script is now stored and documented in My Oracle Support Using the DBhang Script to Gather Debug Data about Messaging Server (Doc ID 1126356.1).

The `pkgapp` script is now stored and documented in My Oracle Support Using Pkgapp to gather libraries to debug core/gcore/crash files. (Doc ID 1274584.1).

To Run the dbhang Script

Note
The dbhang script (version 3) now uses pkginfo to find the *server-root* directory and from that gathers other necessary information from configutil, thereby avoiding the need to edit the script. You can still use command-line switches to override these defaults. A side-effect of this enhancement is that the new version works only on Messaging Server 6.0 and later releases. If you are running iPlanet Messaging Server 5.2, continue to use dbhang version 2.10.

The dbhang script gathers data about problems where the Message Store database is the source of the hang. The dbhang script is not intended to be an all-purpose data gathering script for Messaging Server.

Steps 1, 2, and 3 should only be necessary with dbhang v2 on iMS 5.2.

1. Customize the dbhang script to reflect the specificities of your Messaging Server installation. Use an editor to modify the script then change the SRVROOT, INST, and MAILUSER variables in the top part of the script.
2. Using the configutil command, check if the store.dbtmpdir parameter is set. If it is set, change DBTMPDIR to the same value as store.dbtmpdir, otherwise it should be $MBOXLIST.
3. Leave the following variables set to their default values unless otherwise instructed by Oracle Support:
```
doDBSTAT=1
doMBOXTAR=0
doSHARES=0
doGCORE=0
doPFILES=0
doDISPSTATS=0
doGETCONFIG=0
doPMFCTL=0
dashN=""
```
You should not need to change anything below this comment:
```
#
# you should not need to edit anything below here
#
```

   Note
   Occasionally, the dbhang script cannot find the log files in their expected location, so you might need to change this as well.

4. Perform a test run of the dbhang script after you have made your edits. Ensure that the script runs without errors.
5. Execute the dbhang script and collect the data when you have a problem.

To Run the pkgapp Script

This script packages an executable and all of its shared libraries into one compressed tar file given the PID of the application and optionally the name of the core file to be opened. The files are stripped of their directory paths and are stored under a relative directory named app/ with their name only, allowing them to be unpacked in one directory.

On Solaris 9 OS or greater, the list of files is derived from the core file rather than the process image if it is specified. You still must provide the PID of the running application to assist in path resolution.

Two scripts are created to facilitate opening the core file when the tar file is unpacked:

- opencore. This is the script to be executed once unpacked. It sets the name of the core file and

the linker path to use the `app/` subdirectory and then invokes `dbx` with the `dbxrc` file as the argument.
- `dbxrc`. This script contains the `dbx` initialization commands to open the core file.

1. Copy the script to a temporary directory on the system where Portal Server is installed.
2. Become `superuser`.
3. Execute the `pkgapp` script in one of the following three ways:
    - `./pkgapp` *pid-of-running-application corefile*
    - `./pkgapp` *pid-of-the-running-application* (The *pkgapp* scripts prompts for the *corefile* name.)
    - `./pkgapp` *core file*

Reporting Problems

Use the following email aliases to report problems with this document and its associated scripts:

- To provide feedback: gdd-feedback@sun.com
- To report problems: gdd-issue-tracker@sun.com

Accessing Oracle Resources Online

The http://downloads.oracle.com web site enables you to access Oracle technical documentation online. You can browse the archive or search for a specific book title or subject. Books are available as online files in PDF and HTML formats. Both formats are readable by assistive technologies for users with disabilities.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

> **Note**
> Oracle is not responsible for the availability of third-party web sites mentioned in this document. Oracle does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Oracle will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Chapter 64. Upgrading from Messaging Server 5.2 to Sun Java System Messaging Server

8th April 2009: This page was previously published on the Communications Suite Hub, which has been decommissioned.

Upgrading from Messaging Server 5.2 to Sun Java System Messaging Server

July 2007
This article was previously published as **Chapter 2** of the **Sun Java Systems Messaging Server 6.3 Administration Guide**. This article describes how to upgrade from Messaging Server 5.2 to Messaging Server 6.3. Direct upgrade from Messaging Server 5.2 to 6.3 is not certified, but you can upgrade to 6.2 using procedures described here, then upgrade to 6.3 using the procedures described in the Sun Java Communications Suite 5 Upgrade Guide.

> **Note**
> Depending on the complexity of your deployment, upgrading from Messaging Server 5.2 to 6.3 may not be a trivial task. Sun Professional Services can also assist in upgrading Messaging Server.

Before You Begin

Prior to performing the upgrade, ensure the following:

- The 6.2 version of Messaging Server is installed and configured on either the same or a different system than the Messaging Server 5.2 system.

> **Note**
> Unlike previous versions of Messaging Server, you cannot upgrade your existing Messaging Server without first installing and configuring the 6.2 version of Messaging Server.

Also, you cannot use this upgrade program with Messaging Server versions older than version 5.2. Therefore, you must first migrate or upgrade to Messaging Server 5.2, install the current version of Messaging Server and then run this upgrade program. See the iPlanet Messaging Server 5.2 Migration Guide for more information on migrating to Messaging Server 5.2.

- Existing Messaging Server 5.2 installations are configured with MTA Direct LDAP Lookup, not with imsimta dirsync.
- In addition, Messaging Server does not support multiple instances in the 6.2 release.

Overview of the Upgrade Process

The following topics outline the steps to upgrade from Messaging Server 5.2 to the current version of Messaging Server.

This section describes how special upgrade files are created in order to update the configuration on your Messaging Server:

About Upgrade Files

Prior to running an upgrade utility to move from Messaging Server 5.2 to 6, you first need to run the `UpgradeMsg5toMsg6.pl` Perl script (located in *msg_svr_base*`/sbin`).

`UpgradeMsg5toMsg6.pl` compares your 5.2 configuration files with your Messaging Server 6 configuration files and creates two sets of files for each configuration file: *.CHANGES files and *.MERGED files.

The *.CHANGES files and *.MERGED files are generated in the work space directory, `/var/tmp/UpgradeMsg5toMsg6.ScratchDir`.

The *.CHANGES files show critical configuration file differences between Messaging Server 5.2 and the current version of Messaging Server. These files highlight the configuration entities that are only found in Messaging Server, the configuration entities from Messaging Server 5.2 that are obsolete in this current version of Messaging Server, and the configuration entities that are only found in the Messaging Server 5.2. Not all *.CHANGES files will show differences between the versions of configuration files, and not all configuration files will generate *.CHANGES files.

The *.MERGED files are a consolidation of Messaging Server 5.2 and the current version of Messaging Server configuration values and settings. In general, configuration parameter values from Messaging Server 5.2 are retained over the current version of Messaging Server if:

- There is no default value in the current version of Messaging Server, or
- The value specified in your 5.2 configuration is not a default setting.

The following table lists the configuration files that generate *.MERGED or *.CHANGES files.

Messaging Server Configuration Files that Generate *.MERGED or *.CHANGES files

Configuration Information	Description	Generates *.MERGED File	Generates *.CHANGES File
job_controller.cnf	Job Controller File	X	X
conversions	Conversions File	X	
channeloption, where _channel is an SMTP channel	SMTP channel option files	X	
native_option	Native channel option file (exception to *channel* _option)	X	X
channel headers.opt,where _channel is an SMTP channel	Header option files	X	
dispatcher.cnf	Dispatcher File	X	X
imta_tailor	Tailor File	X	X
option.dat	Global MTA Option File	X	X
aliases	Aliases File	X	
imta.cnf	MTA Configuration File. Only the include references (like file directory locations) are changed. Rewrite rule and channel settings are retained from your 5.2 configuration. To include LMTP in your imta.cnf, copy the LMTP information from your Messaging Server 6 imta.cnf file.	X	In some instances, a *.CHANGES file may be generated.
mappings	Mappings File	X	
mappings.locale	Localized Mappings File	X	
internet.rules	Internet Rule Configuration File	X	
backup-groups.conf	Backup Group Definitions	X	X
configutil	Changes of configuration parameters in local.conf and msg.conf configuration files.		X

To Run the UpgradeMsg5toMsg6.pl **Perl Script**

To run the `UpgradeMsg5toMsg6.pl` to create sets of files by which you'll be able to update your configuration, follow these steps:

Before You Begin

Both your 5.2 and current version of Messaging Server can be running at this point.

If your Messaging Server 5.2 and 6 versions are on the same machine, start with Step 2.

1. If your Messaging Server 5.2 and 6 versions are not on the same machine, transfer, extract and copy the Messaging Server 5.2 *server-root* directory to the current version of Messaging Server.If your server versions are installed on the same machine, you can skip this step.
   If your Message Store is too large to transfer from one system to another, you can transfer just the essential portions of the server instance to the new system. There are comments inside of the `UpgradeMsg5toMsg6.pl` which cover this in detail.

You don't have to copy the Messaging Server 5.2 store data to the current Messaging Server system, however, you must ensure that the Messaging Server 5.2mboxlist directory is accessible during the upgrade process.

2. Run the `UpgradeMsg5toMsg6.pl` upgrade script. By default, this script is located in *msg_svr_base*/sbin.
   Run the script against the msg-*instance* of 5.2 version and the *msg_svr_base* of the current version of Messaging Server. For example:

   ```
   perl UpgradeMsg5toMsg6.pl /usr/sunone/server5/msg-budgie
   /opt/SUNWmsgsr
   ```

   where `/usr/sunone/server5/msg-budgie` is the msg-*instance* of the 5.2 Messaging Server and `/opt/SUNWmsgsr` is the *msg_svr_base* of the current version of Messaging Server.
   The process creates *.MERGED and *.CHANGES files (as described in About Upgrade Files).

3. Carefully review the *.MERGED files to determine if you need to adjust the settings.If you don't want to use the suggested recommendations, you must manually adjust the settings.

   Note -
   This utility does not update the Messenger Express customization files. Therefore, you need to manually change these files in order to keep the relevant information from Messaging Server 5.2 and add any new information from the current version of Messaging Server installation.

Running the Upgrade Utility

This section describes the `do_the_upgrade.sh utility` (located in `/var/tmp/UpgradeMsg5toMsg6.ScratchDir`), a shell script that is made up of four sub-scripts. The following topics are covered in this section:

- Overview of the Upgrade Utility (`do_the_upgrade.sh`)
- MTA Configuration (`make_mta_config_changes.sh`)
- configutil Parameters (`make_configutil_changes.sh`)
- Backup Configuration (`make_backup_config_changes.sh`)
- mboxlist Database (`make_mboxlistdb_changes.sh`)

Overview of the Upgrade Utility

The `do_the_upgrade.sh` utility is made up of four shell scripts that, with your *.MERGED files, update the configuration and file directory locations of your MTA configuration, your configutil parameters, backup parameters, and your mboxlist database in your current version of Messaging Server system.

You can either run the `do_the_upgrade.sh` utility, or you can individually run one or more of the scripts that make up the `do_the_upgrade.sh` utility (`make_mta_config_changes.sh`, `make_configutil_changes.sh`, `make_backup_config_changes.sh`, and `make_mboxlistdb_changes.sh`).

If you want to upgrade an MTA relay machine from Messaging Server 5.2 to current version of Messaging Server, you only need to run the `make_mta_config_changes.sh` and the `make_backup_config_changes.sh` (described in Backup Configuration).

When executing either the `do_the_upgrade.sh` utility or any of the sub-scripts, be sure that neither Messaging Server 5.2 nor current version of Messaging Server is up and running.

To Run the do_the_upgrade.sh Utility

1. Shut down both the 5.2 and the current version of Messaging Servers.
2. Run the utility:

```
# sh /var/tmp/UpgradeMsg5toMsg6.ScratchDir/do_the_upgrade.sh
```

After running the `do_the_upgrade.sh` script, you can either continue to reference your 5.2 partition paths (though you will not be able to remove your Messaging Server 5.2 *server-root* directory) or you can manually move the 5.2 store partitions to the appropriate current version of Messaging Server directory location. You should perform this step prior to restarting Messaging Server.

MTA Configuration

The MTA upgrade configuration sub-script that makes up of part of the `do_the_upgrade.sh` utility is called `make_mta_config_changes.sh` (located in `/var/tmp/UpgradeMsg5toMsg6.ScratchDir`).

The `make_mta_config_changes.sh` script backs up, renames, and moves the *.MERGED server configuration files to their original names and locations within the current version of Messaging Server file directory structure.

Once the script has finished renaming and moving the files, it automatically runs the `imsimta cnbuild` command to recompile the MTA configuration.

Note
If you want to upgrade an MTA relay machine from Messaging Server 5.2 to the current version of Messaging Server, you only need to run the `make_mta_config_changes.sh` and the `make_backup_config_changes.sh` (described in Backup Configuration).

configutil Parameters

The `configutil` upgrade configuration sub-script that makes up part of the do_the_upgrade.sh utility is called `make_configutil_changes.sh` script (located in `/var/tmp/UpgradeMsg5toMsg6.ScratchDir`).

The `make_configutil_changes.sh` script incorporates new or updated parameters in the `msg.conf` and `local.conf` files. If default values are not specified in configutil parameters in the current version of Messaging Server, any Messaging Server 5.2 values are carried forward to the current version of Messaging Server.

Backup Configuration

The backup upgrade configuration sub-script that makes up part of the `do_the_upgrade.sh` utility is called `make_backup_config_changes.sh` script (located in `/var/tmp/UpgradeMsg5toMsg6.ScratchDir`).

The `make_backup_config_changes.sh` script upgrades the configuration of your backup services such as those in your `backup-groups.conf` file.

mboxlist Database

The `mboxlist` database upgrade configuration sub-script that makes up part of the `do_the_upgrade.sh` utility is called `make_mboxlistdb_changes.sh` script (located in `/var/tmp/UpgradeMsg5toMsg6.ScratchDir`).

The `make_mboxlistdb_changes.sh` script transfers and upgrades your 5.2 mboxlist database and upgrades it to the current version of Messaging Server directory structure. The script copies the four *.db

files (folder.db, quota.db, peruser.db, and subscr.db) from *server-root*/msg-*instance*/store/mboxlist on your Messaging Server 5.2 system to *msg_svr_base*/data/store/mboxlist on your current version of Messaging Server system.

Migrating User Mailboxes (Optional)

If you need to migrate your user mailboxes, refer to Migrating Mailboxes to a New System.