

Oracle® Communications Messaging Server
Unified Configuration System Administrator's Guide
Release 8.0

July 2015

ORACLE®

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

Copyright © 2007, 2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1. Overview of Messaging Server Unified Configuration	5
2. About MTA Services and Unified Configuration	17
3. Administering Event Notification Service in Messaging Server for Unified Configuration	58
4. BURL Support for SMTP SUBMIT in Unified Configuration	60
5. Configuring and Administering Multiplexor Services in Unified Configuration	66
6. Configuring General Messaging Capabilities in Unified Configuration	78
7. Configuring IMAP IDLE in Unified Configuration	92
8. Configuring Messaging Server for High Availability in Unified Configuration	95
9. Configuring Rewrite Rules in Unified Configuration	98
10. Integrating Spam and Virus Filtering Programs Into Oracle Communications Messaging Server in Unified Configuration	122
11. JMQ Notification in Unified Configuration	170
Configuring a JMQ Notification Service in Unified Configuration (Tasks and Examples)	172
Enabling JMQ Notification in Unified Configuration (Example)	182
JMQ Notification Messages and Properties in Unified Configuration	186
JMQ Notification Overview in Unified Configuration	195
12. LMTP Delivery in Unified Configuration	199
13. Mail Filtering and Access Control in Unified Configuration	212
14. Managing Logging in Unified Configuration	252
15. Message Store and Mailbox Management in Unified Configuration	299
Administering Very Large Mailboxes in Unified Configuration	301
Backing Up and Restoring the Message Store in Unified Configuration	305
Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS in Unified Configuration	316
Configuring Message Expiration in Unified Configuration (Tasks)	320
Configuring POP, IMAP, and HTTP Services in Unified Configuration	325
Handling Message Store Overload in Unified Configuration	339
Managing Message Store Partitions and Adding Storage in Unified Configuration	341
Managing Message Store Quotas in Unified Configuration	344
Managing Message Types in the Message Store in Unified Configuration	349
Managing Shared Folders in Unified Configuration	354
Message Store Administration in Unified Configuration	362
Message Store Architecture and Concepts in Unified Configuration	363
Message Store Automatic Recovery On Startup in Unified Configuration	364
Message Store Maintenance Queue in Unified Configuration	367
Message Store Message Types Overview in Unified Configuration	370
Message Store Quota (Overview) in Unified Configuration	373
Migrating Mailboxes to a New System in Unified Configuration	378
Monitoring Disk Space in Unified Configuration	387
Protecting Mailboxes from Deletion or Renaming (Unified Configuration)	390
Reducing Message Store Size Due to Duplicate Storage in Unified Configuration	391
Specifying Administrator Access to the Message Store in Unified Configuration	395
Troubleshooting the Message Store in Unified Configuration	397
Valid Message Store UIDs and Folder Names in Unified Configuration	402
16. Messaging Server Lemonade Profile 1 Support in Unified Configuration	404
17. Triggering Effects From Transaction Logging. The LOG_ACTION Mapping Table	409
18. Messaging Server Unified Configuration Command-line Utilities	420
19. Monitoring Messaging Server in Unified Configuration	430
Monitoring the MTA in Unified Configuration	441
20. MTA Address Translation and Routing in Unified Configuration	444
21. MTA Concepts in Unified Configuration	468
22. Security and Access Control in Unified Configuration	480
Configuring Authentication Mechanisms in Messaging Server Unified Configuration	481
Configuring Client Access to POP, IMAP, and HTTP Services in Unified Configuration	485

Configuring Encryption and Certificate-Based Authentication in Unified Configuration	492
User and Group Directory Lookups Over SSL in Unified Configuration	500
23. Short Message Service (SMS) in Unified Configuration	501
Configuring Messaging Server for One-Way SMS in Unified Configuration	562
Configuring Messaging Server for Two-Way SMS in Unified Configuration	564
24. SNMP Support in Unified Configuration	567
25. Troubleshooting the MTA in Unified Configuration	582
26. Tuning the mboxlist Database Cache in Unified Configuration	607
27. Using and Configuring MeterMaid for Access Control	618
28. Implementing Greylisting by Using MeterMaid	627
29. Using Predefined Channels in Unified Configuration	641
30. Using Role-Based Access Control in Messaging Server Unified Configuration	672
31. Using the iSchedule Channel to Handle iMIP Messages	675
32. Vacation Automatic Message Reply in Unified Configuration	684
33. Performance Tuning DNS Realtime BlockLists (RBL) Lookups	690
34. Protecting Against Spammers who Compromise Messaging Server User Accounts	697
35. Rate-limiting Email	705
36. Setting Up and Managing Messaging Server Security	711
Messaging Server NFS Guidelines and Requirements	724
37. Setting Up a No Phishing Zone	725
38. Using NetApp Filers with Messaging Server Message Store	727
39. Using IPv6 with Messaging Server	730
40. Veritas Cluster Server Agent Installation	735

Chapter 1. Overview of Messaging Server Unified Configuration

Overview of Oracle Communications Messaging Server Unified Configuration

This information introduces Messaging Server Unified Configuration, describes its capabilities, and provides initial guidelines on how to transition from a legacy configuration. Unified Configuration provides the ability to configure Messaging Server in a way that is much less error-prone and much easier to script and reuse across multiple hosts.

Unified Configuration was introduced in **Messaging Server 7 Update 5**.

Topics:

- [What Is Messaging Server Unified Configuration?](#)
- [Unified Configuration Files](#)
- [Enabling Unified Configuration in Messaging Server](#)
- [Understanding Unified Configuration Limitations](#)
- [Using the Repository of Previous Configurations](#)
- [Using Legacy Configuration Tools with Unified Configuration](#)
- [About Compiled MTA Configurations](#)
- [Separating Roles and Instances](#)
- [More About Unified Configuration Options](#)
- [Example of Legacy Configuration and Unified Configuration](#)
- [Using Recipes](#)
- [Helpful Commands](#)

What Is Messaging Server Unified Configuration?

Unified Configuration is an improved process to configure and administer Messaging Server. Unlike in legacy configurations, Unified Configuration uses validation to verify configuration accuracy, and employs a single tool to configure the entire Messaging Server configuration (with a few exceptions).

The following table describes how Unified Configuration improves upon issues with legacy configuration.

Legacy Versus Unified Configuration

Legacy Configuration Issue	Unified Configuration Improvement
----------------------------	-----------------------------------

<p>Dealing with many configuration files (with inconsistent formats) and hand-editing them can lead to errors and invalid configurations.</p>	<p>Unified Configuration "unifies" configuration management. One tool, <code>msconfig</code>, administers Messaging Server configuration. Validation checking prevents introducing some configuration errors.</p>
<p>Configuration settings themselves are often complicated and not straight-forward.</p>	<p>Unified Configuration reduces redundancy and host specific-configurations, so that, for example, you can use the same settings for many options among the MMP, MTA, and message store configurations.</p>
<p>When problems arise, there are support challenges due to the many Messaging Server configuration files. Additionally, because passwords are contained in the configuration files, it makes it difficult for customers to just send these files to Oracle Support without first removing the passwords.</p>	<p>Unified Configuration uses only three text files to store configuration data, with most data stored in the <code>config.xml</code> file. Passwords are stored in a separate file, removing the need for customers to edit configuration files before sending to Oracle Support. In addition, Unified Configuration provides an audit trail of configuration changes. The changes (currently the last 100 changes) are actually stored in a repository, referred to as a <i>graveyard</i>. Storing of changes further enables you to restore an entire configuration, and to roll-back and roll-forward between configurations.</p> <p>See Unified Configuration Files for more information.</p>
<p>It is difficult to separate instance-specific details from details shared by functionally similar machines.</p>	<p>Unified Configuration separates tasks for single instances (referred to as <i>instance</i>) of Messaging Server from global tasks for a group (referred to as <i>role</i>) of Messaging Server machines. The intention is that the role contain configuration information suitable for sharing with other hosts that have the same function in the deployment. At present, there is no mechanism to automatically share role configuration.</p> <p>See Separating Roles and Instances for more information.</p>
<p>Customers must create their own procedures and scripts with their own tools to manage a deployment.</p>	<p>New customers can write automation scripts by using the Unified Configuration <i>recipe language</i>. In Messaging Server 7 Update 4 and prior releases, you had to manually automate configuration changes, which overwrote configuration files, potentially removed necessary files, and could introduce syntax errors. Also, necessary machine-specific variations in the configuration files made this sort of automation fairly complex. The recipe language introduces the ability to robustly change a configuration in a reproducible fashion in a way that can be sensitive to what was previously in the configuration. When you use recipes, you are able to make use of the Unified Configuration history and administrative undo features. In addition, Oracle can use the recipe language to automate configuration changes that are otherwise complex, interconnected, and require lots of documentation. The <code>SpamAssassin.rcp</code>, <code>HAConfig.rcp</code>, and <code>LMTPSingleSystem.rcp</code> recipes, available in the <code>msg-svr-base/lib/recipes</code> directory, are good examples.</p> <p>See Using Recipes for more information.</p>

Unified Configuration Files

The following table describes the Unified Configuration file names, file management tool, file format, character set, XML schema, ownership, and file permissions. The Unified Configuration files are located in the *configroot* directory, by default, `/var/opt/sun/comms/messaging64/config`.

Unified Configuration File Properties

Configuration File	Description	File Management Tool	File Format	Char Set	File Ownership	Suggested File Permissions
restricted.cnf	Contains protected Messaging Server UID and GID information.	Text editor	option=value	ASCII	root	0644
xpass.xml	Contains obfuscated passwords (BASE64 encoded). This is the only file within Unified Configuration where password information is stored.	msconfig utility	XML 1.0	UTF-8	mailsrv	0600
config.xml	Contains most of the non-password configuration information. In addition, when necessary, you could send this entire file to Oracle Support to help with resolving problems.	msconfig utility	XML 1.0	UTF-8	mailsrv	0640
configlib.xml	Contains static, default mapping tables that are mostly concerned with character sets and language issues, including: - DISPOSITION_LANGUAGE - DOMAIN_DC - LANGUAGE_LOCALES - LDAP_USERS_LANGUAGE LDAP_USERS2_LANGUAGE - NOTIFICATION_LANGUAGE	Managed by Oracle (that is, the file is not to be edited)	XML 1.0	UTF-8	mailsrv	0644

Notes:

- When you perform the initial Messaging Server configuration, the `restricted.cnf` file sets the

UID under which to run Messaging Server. After initial configuration, there should rarely be a need to edit this file. A legacy configuration can also use the `restricted.cnf` file for enhanced security.

- Never edit the `configlib.xml` file. Doing so causes an unsupported configuration.

Note: About MTA Tailor Options

In legacy configuration, you use the MTA tailor file of option settings (`imta_tailor`) to set various MTA installation and operational parameters. In Unified Configuration, the MTA tailor file is obsolete and no longer used. Unified Configuration replaces the MTA tailor options that specified locations of MTA directories or files with rationalized, consistent locations, which are based off the installation main location and located by using the `SERVERROOT` environment variable. Legacy configuration MTA tailor options that set other sorts of MTA operational parameters have typically been replaced with Unified Configuration options of the form `mta.option-name`.

Enabling Unified Configuration in Messaging Server

There are two ways to enable Unified Configuration:

1. **When migrating to Unified Configuration:** Use the `msg-svr-base/bin/configtoxml` program to migrate a legacy configuration to Unified Configuration. When you run `configtoxml`, your old configuration is converted to Unified Configuration.
 - The legacy configuration is saved in the `configroot/legacy-config` directory.
 - If necessary, you can use the `configtoxml -undo` command to restore a saved legacy configuration.
2. **When installing a fresh instance of Messaging Server 7 Update 5 or greater:** For a new Messaging Server instance, run the `configure --xml` command to enable Unified Configuration by default.
 - The presence of a `config.xml` file in the `config` directory indicates that Unified Configuration is enabled.
 - To generate a legacy configuration instead of a Unified Configuration, you can use the `configure -noxml` command.
 - When you perform a fresh installation of Messaging Server 7 Update 5 or greater, and choose to configure a Unified Configuration, you cannot revert that Unified Configuration to a legacy configuration. If, however, you upgrade Messaging Server to Messaging Server 7 Update 5 or greater, and convert to a Unified Configuration (by running the `configtoxml` command), you can revert back to the legacy configuration.
 - A Unified Configuration is more simple than a legacy configuration. In addition, where appropriate, modern default values are established and seldom used features are removed (for example, the `tcp_tas` channel is not present in Unified Configuration).
 - The Messaging Server 7 Update 4 and prior releases configuration files, such as `dispatcher.cnf`, `option.dat`, and so on, are ignored when Unified Configuration is enabled.

To Determine if Unified Configuration Is Deployed

The following example shows how to determine if Unified Configuration is deployed on your system:

```
# cd /opt/sun/comms/messaging64/bin
# ./imsimta version
...
Using /opt/sun/comms/messaging64/config/config.xml
SunOS host2.example.com 5.10 Generic_142901-03 i86pc i386 i86pc
```

In this example, the presence of the `config.xml` file indicates that Unified Configuration has been enabled on this host.

If you are using a compiled configuration and see in that the status is not compiled, you should recompile the configuration. For example:

```
# /opt/sun/comms/messaging64/bin/imsimta version
...
Using /opt/sun/comms/messaging64/config/config.xml (not compiled)
SunOS jet.us.oracle.com 5.10 Generic_147441-09 i86pc i386 i86pc

# /opt/sun/comms/messaging64/bin/imsimta cnbuild
```

For more information, see [Compiling the MTA Configuration](#).

Understanding Unified Configuration Limitations

In general, Unified Configuration has consolidated all the various Messaging Server files. Nevertheless, the current Unified Configuration implementation has a few limitations:

- The channel sieves and channel header trimming option files have not yet been converted to XML.
- Some files, such as localized templates for DSNs and NDNs, might remain in their current format and not be converted to XML.
- The content of the `conversions` file is a mono-block in XML.

Using the Repository of Previous Configurations

The repository of previous configurations, known as the *graveyard*, is stored in the `configroot/old-configs/` directory. The move from current configuration to the graveyard is performed when a new configuration is written to disk. The graveyard maintains the most recent 100 configurations. With the graveyard, you can restore an old configuration by reverting to a previous configuration. Furthermore, you can compare differences between any two configurations, for example, between the active configuration and a previous configuration, or two old configurations.

To List Configurations

- Use the `msconfig history` command to show a list of configurations currently in the graveyard.

To Compare Configurations

- To compare configurations, use the `msconfig differences m_n` command, where *m* and *n* are the numbers of the previous configurations from the `history` command that you want to compare.

Using Legacy Configuration Tools with Unified Configuration

Once Unified Configuration is enabled, legacy configuration tools might work differently than in previous releases. Specifically:

- Scripts that directly alter legacy configuration files do not work in a Unified Configuration.
- The `configutil` command can still set, get, and delete a legacy configuration. Additionally, in Unified Configuration, `configutil` options can also automatically perform:

- Option name translations
- Option value translations
- Option value validations



Note

Use of the `configutil` command is deprecated in Unified Configuration mode. Some configuration changes that were previously possible with the `configutil` command are only possible by using the `msconfig` command, for example, some changes to notification configuration.

- The `mkbackupdir` command, which creates and synchronizes the backup directory with the information in the message store, works with Unified Configuration.
- The `imsimta program` command, which manipulates the program delivery options, does not work with Unified Configuration. Instead, you must use the `msconfig` command.
 - This command issues an error and exits with 1 when Unified Configuration is being used.
- All other utilities only consume options and continue to work with both Unified Configuration and legacy configurations.

About Compiled MTA Configurations

The following changes for compiled MTA configurations are introduced in Messaging Server 7 Update 5:

- The `configure` command no longer generates a compiled configuration by default, for both legacy and Unified Configuration.
- Unified Configuration provides the `-xmlfile=` switch to use test tools on a non-live configuration. (This was previously only possible with a compiled configuration.)
- The `imsimta version` command now shows if a compiled configuration is used or not.

Separating Roles and Instances

Unified Configuration separates tasks for single instances (referred to as *instance*) of Messaging Server from global tasks for a group (referred to as *role*) of Messaging Server machines. The intention is that the role contain configuration information suitable for sharing with other hosts that have the same role in the deployment.



Note

At present, there is no mechanism to automatically share role configuration.

Any configuration option can be an instance setting, a role setting, or both. When the same option is in both the role and instance, the instance value takes precedence. Both the `configure` and `msconfig` commands put a given setting in either the instance or the role based on the likely scope for the option. Normally, you use the default location (instance or role) determined by the `msconfig` command and not explicitly specify one or the other.

The Messaging Server 7 Update 5 initial configuration generates an instance and role, as does migrating from a legacy configuration to Unified Configuration.

More About Unified Configuration Options

This section provides information about password, restricted, and obsolete options.

- [Options That Have Passwords](#)
- [Restricted Options](#)

- [Obsolete Options](#)
- [Option Relationships](#)
- [Unified Configuration Option Names](#)

Options That Have Passwords

The password options have the following characteristics:

- Options can be marked as being a password.
- By default, password values are not displayed.
- The passwords are stored in obfuscated form in the `xpass.xml` file. Because Messaging Server 7 Update 5 stores passwords in a separate file, you do not need to edit configuration files as with previous Messaging Server releases before sharing with Oracle Support.

Restricted Options

Some configuration options are marked "restricted" by Oracle. Additionally, starting with Messaging Server 7 Update 5, the XML schema exposes the entire configuration, so there are no longer any hidden configuration options.

Options may be restricted for several reasons, including:

- The option has complex and subtle consequences and would cause harm in all but a very few rare circumstances.
- The option might be a legacy option that should not be used in new systems.
- The option might be a placeholder for a feature that has not yet been implemented.

Restricted options have the following characteristics:

- The `msconfig` command displays a warning when you attempt to set a restricted option. In addition, the `msconfig` command requires an extra step to actually set the restricted option.
- The restriction is noted within the configuration file itself, which helps you to be aware of any special circumstances. For example:

```
<delimiter_char v="127" xannotation="RESTRICTED USAGE OPTION: user  
remark" xauthor="dcn@example.com" xmtime="2010-05-12T17:42:19-08:00" />
```

The `user remark` text is any optional remark added by the administrator when the configuration was updated. The "RESTRICTED USAGE OPTION" is text inserted by Oracle into the remark field when the option is restricted.



Caution

If you set a restricted option without being advised to do so by Oracle Support, your configuration is considered unsupported by Oracle.

Obsolete Options

When an option has been marked by Oracle as being obsolete, the configuration no longer uses it. However, you cannot remove it from the XML Schema as that would make existing configurations invalid.

When marked as obsolete, the option:

- Remains in the XML Schema
- Can no longer be set or changed
- Can only be deleted from a configuration

Option Relationships

Unified Configuration enables some relationships between options to be expressed so that when a particular option is set, other unnecessary options can be automatically removed. For example, if you set the `mx` keyword on a channel (for MX mail forwarding records), any of the `nomx`, `randommx`, and other related "mx" keywords are removed. In addition, Unified Configuration uses the concept of default relationships to help with configuration. For example, option X and option Y might have a default relationship such that when X is not set, the value is taken from Y; or when Y is not set, then Y's default value is *value*. Furthermore, Unified Configuration has the capability to know in which release an option became available and warn when a certain configuration is not release-suitable. In general, option relationships help to reduce configuration mistakes.

Unified Configuration Option Names

Unified Configuration uses a "unified" option naming convention that is reminiscent of Messaging Server 7 Update 4 and prior releases `configutil` option names.

In general, this option naming convention uses the following structure:

```
[role.]group[.sub-group].sub-group].option  
[instance.]group[.sub-group].sub-group].option
```

The following example shows a *group.sub-group.option* convention:

```
imap.logfile.flushinterval
```

In this example, `imap` is the group, `logfile` is the sub-group, and `flushinterval` is the option.

This example shows a *group.option* convention:

```
mta.mm_debug
```

In this example, `mta` is the group and `mm_debug` is the option.

Characteristics about option names to keep in mind:

- Many groups only appear once (for example, `imap` and `pop`).
- Some groups may appear many times.
For example:

```
channel  
mapping  
sectoken  
alias  
task
```

- The group or sub-group can include a `:name` portion used for "named" groups.
For example:

```
channel:tcp_local.slave_debug  
partition:primary.path
```

Characteristics about instances and roles to keep in mind:

- An option in the "instance" overrides the same option in the "role." For example, IMAP is effectively disabled by this configuration:

```
instance.imap.enable = 0
role.imap.enable = 1
```

- There actually is no option called `imap.enable`. It is either `role.imap.enable` or `instance.imap.enable`.
- When setting options, you typically do not specify either "role" or "instance." The `msconfig` command applies heuristics to determine whether "role" or "instance" applies. Here is a sample, basic `config.xml` file that shows how the configuration uses instance and role:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xconfig ...>
<role name="store">
<base>
<defaultdomain v="siroe.com"/>      role.base.defaultdomain
</base>
<imap>
<enable v="1"/>      role.imap.enable
<numprocesses v="2"/>      role.imap.numprocesses
</imap>
<mapping name="ABC">
<rule pattern="x*y" template="$N"/>      role.mapping:ABC.rule
<rule pattern="" template="$Y"/>      *role.mapping:ABC.rule
</mapping>
</role>
<instance name="ims" roleref="store">
<base>
<hostname v="wassonite.siroe.com"/>      instance.base.hostname
</base>
<mta>
<mm_debug v="5"/>      instance.mta.mm_debug
</mta>
</instance>
</xconfig>
```

In the preceding example, some option names that you would see upon listing them with the `msconfig show` command are displayed in bold. Also, you can see that the default domain (`defaultdomain`), number of IMAP processes (`numprocesses`), and mappings (`mapping name`) have been defined for the store role; and that the host name (`hostname`) and MTA logging debug level (`mm_debug`) have been set for the store instance.



Tip

Use the `configutil -H` command to translate the legacy `configutil` option names to Unified Configuration names. For example:

```
# configutil -H -o logfile.imap.expirytime
Configuration option: logfile.imap.expirytime
Unified Config Name: imap.logfile.expirytime
```

Example of Legacy Configuration and Unified Configuration

Unified Configuration greatly simplifies the configuration process, as shown in this example of configuring the SMTP server for debugging.

In a legacy configuration, you need to perform the following steps:

1. Edit the `imta.cnf` file and modify the `tcp_local` channel entry:

```
tcp_local identnonnumeric inner loopcheck maysaslserver
maytlsserver mx \
pool SMTP_POOL remotehost saslswitchchannel tcp_auth smtp
sourcespamfilter1 \
switchchannel master_debug
tcp_local-daemon
```

2. Edit the `option.dat` file and add the following option:

```
mm_debug=3
```

3. Edit (or create if it does not exist) the `tcp_local_option` file and add the following option:

```
trace_level=2
```

4. Check permissions on all files, especially the `tcp_local_option` file.

In Unified Configuration, the equivalent steps to the preceding task are the following:

```
% msconfig
msconfig> set channel:tcp_local.slave_debug
msconfig# set mm_debug 3
msconfig# set channel:tcp_local.options.trace_level 2
msconfig# write
```

Using Recipes

You use recipe files, which are expressed by using a programming language, to automate configuration tasks, typically by scripting them. Messaging Server 7 Update 5 recipes are located in the `msg-svr-base/lib/recipes` directory. The primary inspiration for the recipe language is the Icon programming language designed by Ralph Griswald. As such, it supports C-like expressions, operators, and assignments, Sieve-like conditionals, and loops. The available data types are integers, strings, and lists.

Recipes typically operate in three phases. First, a number of checks are done to make sure the right conditions exist for the recipe to be effective. Next the recipe asks a number of questions to determine exactly what changes should be made. Finally, the recipe implements the requested changes. Note that while this is the typical ordering, recipes are not constrained to use it and may use other approaches if appropriate.

By using the recipe language, you can more easily script complex configuration changes. For more information on writing recipes, see the help text for the recipe language by typing `msconfig -help` and choosing help for the `Recipe_language` topic, or refer to [Recipe Language](#).

To Run a Recipe

To run a recipe, type the following command:

```
msconfig run <recipe_name>
```

Helpful Commands

This section provides some helpful commands to get started with Unified Configuration.

- [To Show Settings](#)
- [To Get Help](#)

To Show Settings

- Use the `msconfig show` command to display current settings. For example, to show all currently enabled options:

```
# /opt/sun/comms/messaging64/bin/msconfig show *enable
role.watcher.enable = 1
role.schedule.enable = 1
role.store.enable = 1
role.store.purge.enable = 1
role.imap.enable = 1
role.pop.enable = 1
role.mta.enable = 1
role.dispatcher.service:SMTP.enable = 1
role.dispatcher.service:SMTP_SUBMIT.enable = 1
role.mmp.enable = 0
role.ens.enable = 1
role.http.enable = 1
```

To Get Help

- Use the `msconfig help` command.

Chapter 2. About MTA Services and Unified Configuration

About MTA Services and Unified Configuration

This information describes general MTA services and configuration in Unified Configuration.

Unified Configuration was introduced in **Messaging Server 7 Update 5**.

Topics:

- [MTA Configuration Overview](#)
- [Compiling the MTA Configuration](#)
- [Channels and Rewrite Rules](#)
- [To Convert Addresses from an Internal Form to a Public Form](#)
- [Controlling Delivery Status Notification Messages](#)
- [Controlling Message Disposition Notifications](#)
- [Optimizing MTA Performance](#)

MTA Configuration Overview

In a legacy configuration, you manage the MTA configuration by editing various text files. In Unified Configuration, you manage the MTA configuration by using the `msconfig` command. The Unified Configuration stores the MTA configuration in a single file, `config.xml` (for the most part). The `msconfig` command performs syntax validation on option names and values as well as the Unified Configuration structure to a limited degree. For more information on the syntax and options for the `msconfig` command, see [msconfig Command](#).



Caution

Only edit your Unified Configuration by running the `msconfig` command. This saves old configurations and allows for rollback. Do not hand-edit any of the Unified Configuration files. Oracle Support may occasionally edit these files to work around any issues pertaining to `msconfig`.

The following table provides a comparison of how the MTA configuration is managed in legacy and Unified Configuration.

MTA Configuration: Legacy Versus Unified Configuration

Legacy Configuration Files	Unified Configuration Method
mappings file	Running <code>msconfig edit mappings</code> loads mappings in legacy format in the administrator's chosen editor.
imta.cnf file	Running <code>msconfig edit channels</code> loads channel blocks in legacy format in the administrator's chosen editor. Running <code>msconfig edit rewrite</code> loads rewrite rules in legacy format in the administrator's chosen editor.
option.dat file	Get or set options directly by running <code>msconfig</code> .
job_controller.cnf file	Get or set options directly by running <code>msconfig</code> ; most options require <code>job_controller</code> prefix.
dispatcher.cnf file	Get or set dispatcher options; most require a <code>dispatcher.</code> prefix, service-specific settings are in a <code>service:name</code> group, such as <code>service:SMTP.tcp_ports</code> .
conversions file	Run <code>msconfig edit conversions</code> .
aliases file	Run <code>msconfig edit aliases</code> .

Compiling the MTA Configuration

Changes for Compiled MTA Configurations

The following changes for compiled MTA configurations were introduced in Messaging Server 7 Update 5:

- The `configure` command no longer generates a compiled configuration by default, for both legacy and Unified Configuration.
- Unified Configuration provides the `-xmlfile=` switch to use test tools on a non-live configuration. (This was previously only possible with a compiled configuration.)
- The `imsimta version` command now shows if a compiled configuration is used or not.

Recompiling the MTA Configuration

Whenever you make a change to the MTA configuration, such as to mappings, rewrite rules, channel blocks, aliases, and so on, you must recompile the configuration, if you are using a compiled configuration. This compiles the configuration files into a single image in shared memory.

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

Using a compiled configuration enables you to be able to test configuration changes more conveniently because 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 reading the `config.xml` 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. If you make changes to rewrite rules, channel blocks, mappings, aliases, conversions, or job controller options, then you should issue an `imsimta cnbuild` command followed by an `imsimta restart smtp` command. If you make changes to

dispatcher options, you need to do an `imsimta restart dispatcher` command. If you make changes to options 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 options 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 "MTA Commands" in *Messaging Server Administration Reference* for details on these commands.)

Other instances where you must restart the job controller:

- Changing the job controller configuration.
- Adding or changing use of the channel options `pool`, `maxjobs`, `master`, `slave`, `single`, `single_sys`, or `multiple`. Adding or changing a `threaddepth` channel option can be dealt with instead by running the `imsimta cache -change -thread_depth=...` command.
- 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 channels. (Changes to mappings 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 mappings or to databases can often be handled by running the `imsimta reload` command, 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.

Any changes (such as additions or changes to options on channel definitions) also require running the `imsimta cnbuild` command. 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.

Channels and Rewrite Rules

In Unified Configuration, you add, modify, or delete channel definitions and rewrite rules as follows:

- To use editor mode: `msconfig edit channels` and `msconfig edit rewrite`
- To make changes to individual options: `msconfig set option value`
- To delete individual options: `msconfig unset option value`

The channel associated with a rewritten destination address becomes the destination channel. The system will typically work well using the default configuration.

This section provides a brief introduction to the MTA configuration for channels and rewrite rules. For details about configuring the rewrite rules and channel definitions, see [Configuring Rewrite Rules in Unified Configuration](#) and http://msg.wikidoc.info/index.php/Channel_configuration.

By modifying the MTA configuration, 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 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 MTA configuration contains domain rewriting rules and channel definitions. The channel definitions are collectively referred to as the channel table. An individual channel definition forms a channel block.

The following legacy example configuration 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. In the legacy configuration, the rewrite rules appear in the upper half followed by the channel definitions in the lower half.



Note

This example shows both rewrite rules and channel definitions together. In practice, in Unified Configuration, you would have to use the separate `msconfig edit rewrite` and `msconfig edit channels` commands to view the information.

```
! 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
1      (4)
local-host

a_channel defragment charset7 usascii      (5)
a-daemon

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

The key items (labeled with numbers, enclosed in parentheses) in the preceding example 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. The exclamation point (!) is translated to the `<annotation>` element in the `config.xml` file.
2. The rewrite rules appear in the first half. 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 `1` channel. Blank lines then separate each channel block from one another. (An exception is the `defaults` channel, which can appear before the `1` channel).
5. A typical channel definition consists of a channel name (`a_channel`), some options which define the configuration of a channel (`defragment charset7 usascii`), and a routing system (`a-daemon`), which is also called a *channel tag*.
6. In Unified Configuration, the concept of including the contents of other files no longer applies. The entire configuration needs to reside in the `config.xml` file with all mappings present. Note that when you migrate a legacy configuration to Unified Configuration by using the `configtoxml`

migration tool, it pulls in all external mapping files.

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 in Unified Configuration](#) for more information on the MTA configuration file.



Note

Whenever changes are made to the MTA configuration for channels and rewrite rules, the MTA configuration must be recompiled if running a compiled configuration. See [Compiling the MTA Configuration](#).

To Add the `master_debug` Channel Option to the `tcp_local` Channel

The following is a Unified Configuration recipe example that shows how to add the `master_debug` channel option to the `tcp_local` channel, or to add a "reject this IP address" rule to the `PORT_ACCESS` mapping.

```
set_option("channel:tcp_local.master_debug");  
  
append_mapping("PORT_ACCESS", ["<pattern>", "<template>"]);
```

Mappings

In Unified Configuration, you add, modify, or delete mappings with the `msconfig` command as follows:

- To use editor mode: `msconfig edit mappings`
- To add a rule, enter the `msconfig` interactive mode: `msconfig> set mapping:name.rule pattern template`
- To delete individual options: `msconfig unset option value`

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 string is compared. The second (right-hand) column supplies the *template* by which the input string is mapped to an output string.

In Unified Configuration, MTA mapping tables are represented by mapping XML elements and they are referenced from within the `msconfig` utility as:

```
mapping:mapping-name
```

Wildcard capabilities are provided, as well as multistep and iterative mapping methods.

The mappings are incorporated into the compiled configuration as part of the reloadable section (see [Compiling the MTA Configuration](#)). Whenever changes are made to mappings, and you are using a compiled configuration, the MTA configuration must be recompiled.

The following table lists the MTA mapping tables.

Messaging Server Mapping Tables

Mapping Table	Description
AUTH_REWRITE	Used with the <code>authrewrite</code> option to modify header and envelope addresses using addressing information obtained from authentication operations (SASL). See http://msg.wikidoc.info/index.php/Configutil_Reference .
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 http://msg.wikidoc.info/index.php/Configutil_Reference .
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 <code>tcp_smtp_server</code> when MAIL FROM is received. Use this element 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 element. 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 elements. Used by <code>tcp_smtp_server</code> 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 . Note: NOTIFICATION_LANGUAGE elements are contained in the <code>configlib.xml</code> file.
ORIG_MAIL_ACCESS	Same as MAIL_ACCESS except using the value of the <i>original</i> 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 <i>original</i> 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 http://msg.wikidoc.info/index.php/Configutil_Reference .
PORT_ACCESS	Block incoming connections based on source and destination IP address and TCP port number. Used by <code>dispatcher</code> immediately after accepting the TCP connection. Used by <code>tcp_smtp_server</code> when connection is handed off from <code>dispatcher</code> . 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 <code>tcp_smtp_server</code> when RCPT TO is received. See Controlling Access with Mapping Tables .
<i>SMS_Channel</i> _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 .

Mappings Format

Mappings consist 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.

When you view mappings by using the `msconfig edit mappings` command, you see something that resembles the following:

```
<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>

    .
    .
    .
```

The actual schema in the config.xml looks like the following:


```

<mapping name="table1_name">
  <rule pattern="pattern1-1" template="template1-1"/>
  <rule pattern="pattern1-2" template="template1-2"/>
  <rule pattern="pattern1-3" template="template1-3"/>
  .
  .
  .
  <rule pattern="pattern1-n" template="template1-n">
</mapping>
<mapping name="table2_name">
  <rule pattern="pattern2-1" template="template2-1"/>
  <rule pattern="pattern2-2" template="template2-2"/>
  <rule pattern="pattern2-3" template="template2-3"/>
  .
  .
  .
  <rule pattern="pattern2-n" template="template2-n"/>
</mapping>
<mapping name="table3_name">
  .
  .
  .
</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 when creating mappings configurations.

Including Other Files in the Mappings File

In Unified Configuration, the concept of being able to include other files in the mapping configuration no longer applies. When you migrate a legacy configuration to Unified Configuration, if your configuration has include files references, they are merged into the new Unified Configuration.

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 <i>n</i> th 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
<code>\$n</code>	The <i>n</i> th wildcard field as counted from left to right starting from 0.
<code>\$# . . . #</code>	Sequence number substitution.

<code>\$....[</code>	URL lookup; substitute in result.
<code>\$. . . </code>	Applies specified mapping table to supplied string.
<code>\${...}</code>	General database substitution.
<code>}\$domain, attribute{</code>	<p>Adds the capability to access per-domain attributes. <i>domain</i> is the domain in question and <i>attribute</i> 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.</p> <p><i>attributes</i> can be domain LDAP attributes or the special attributes defined below:</p> <p><code>_base_dn_</code> - The base DN for user entries in the domain <code>_domain_dn_</code> - The DN of the domain entry itself <code>_domain_name_</code> - The name of the domain (as opposed to an alias) <code>_canonical_name_</code> - The canonical name associated with the domain</p>
<code>\$[. . .]</code>	Invokes site-supplied routine; substitute in result.
Metacharacter	Description
<code>\$C</code>	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) .
<code>\$E</code>	Ends the mapping process now; uses the output string from this entry as the final result of the mapping process. <code>+\$1E</code> exits immediately without interpreting the rest of the template. See Processing Control (\$C, \$L, \$R, \$E) .
<code>\$L</code>	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 <code>\$C</code> , <code>\$E</code> , or <code>\$R</code> metacharacter. See Processing Control (\$C, \$L, \$R, \$E) .
<code>\$R</code>	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) .
<code>\$nA</code>	Inserts the <i>n</i> th left character of the current address starting from position 0. The entire address is inserted if <i>n</i> is omitted.
<code>\$nX</code>	Inserts the <i>n</i> th left component of the mailhost starting from 0. The entire mailhost is inserted if <i>n</i> is omitted.
<code>\$?x?</code>	Mapping entry succeeds <i>x</i> percent of the time.
<code>\$</code>	Forces subsequent text to lowercase.
<code>\$^</code>	Forces subsequent text to uppercase.
<code>\$_</code>	Leaves subsequent text in its original case.
<code>\$=</code>	Forces subsequent substituted characters to undergo quoting appropriate for insertion into LDAP search filters.
<code>:\$x</code>	Match only if the specified flag is set.
<code>;\$x</code>	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 (\$ {...})
- 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$%*::*      $l@$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 controls 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 by 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|*          $$?$?25?$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 *restricted.cnf* file, specified by the *noprivuser* option) in order to use such sequence number files.

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 with the *ugldaphost* and *ugldapport* 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*, 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 by 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.

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 Information

In legacy configuration, in addition to the `imta.cnf` file, Messaging Server provides several other configuration files to help you configure MTA services. Unified Configuration uses only three files to store configuration data, with most data stored in the `config.xml` file. The following table summarizes how the legacy configuration files map to Unified Configuration.

Location of Additional MTA Configuration

File	Description	Legacy	Unified Configuration
Aliases	Implements aliases not present in the directory.	<code>msg-svr-base/config/aliases</code>	<code>msg-svr-base/config/config.xml</code> , <code>msg-svr-base/config/xpass.xml</code>
TCP/IP (SMTP) Channel Option	Sets channel-specific options.	<code>msg-svr-base/config/channel_option</code>	<code>msg-svr-base/config/config.xml</code> , <code>msg-svr-base/config/xpass.xml</code>

Conversion	Used by the conversion channel to control message body part conversions.	<i>msg-svr-base</i> /config/conversions	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml
Dispatcher	Sets dispatcher configuration.	<i>msg-svr-base</i> /config/dispatcher.cnf	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml
Job Controller	Sets job controller configuration.	<i>msg-svr-base</i> /config/job_controller.cnf	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml
MTA Main Configuration	Used for address rewriting and routing as well as channel definition.	<i>msg-svr-base</i> /config/imta.cnf	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml
Mappings	Repository of mapping tables/elements.	<i>msg-svr-base</i> /config/mappings	<i>msg-svr-base</i> /config/config.xml <i>msg-svr-base</i> /config/xpass.xml
Configutil		<i>msg-svr-base</i> /config/msg.conf	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml
Option	Sets global MTA options.	<i>msg-svr-base</i> /config/option.dat	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml
Tailor	Specifies locations and some tuning parameters.	<i>msg-svr-base</i> /config/imta_tailor	<i>msg-svr-base</i> /config/config.xml, <i>msg-svr-base</i> /config/xpass.xml, <i>msg-svr-base</i> /config/restricted.cnf
General Lookup Table	General lookup facility is equivalent to the general database. Part of reloadable compiled configuration. File to specify locations and some tuning parameters.	<i>msg-svr-base</i> /config/general.txt	No change
Forward Lookup Table	Lookup facility for To: addresses. Equivalent to forward database. Part of reloadable compiled configuration.	<i>msg-svr-base</i> /config/forward.txt	No change

Reverse Lookup Table	Reverse lookup facility for From: addresses. Equivalent to reverse database. Part of reloadable compiled configuration.	<code>msg-svr-base</code> <code>/config/reverse.txt</code>	No change
----------------------	---	---	-----------



Note

The text databases are separate from product configuration and their behavior is unchanged in Unified Configuration.

Aliases

You can edit the configuration to set aliases not in the directory. In particular, the address for root is a good example. Aliases set are ignored if the same aliases exist in the directory. For more information about aliases and the `aliases` file, see [Aliases](#).

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

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. For complete information on the MTA command-line utilities, see *Messaging Server Administration Reference*.

In Unified Configuration, the `configutil` command is deprecated should not be used for setting options. Use the `msconfig` tool for all configuration editing operations.

SMTP Security and Access Control

For information about SMTP security and access control, see [Mail Filtering and Access Control in Unified Configuration](#).

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 in Unified Configuration](#).

To Convert Addresses from an Internal Form to a Public Form

Addresses can be converted from an internal form to a public, advertised form by 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 in Unified Configuration](#).

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`. 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, 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]NewAddress
```

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.
<i>Flag comparison</i>	<i>Description</i>
\$: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 database files is eventually being deprecated because of the instability it introduces in Messaging Server deployments. 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), 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 database
bit 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 options, 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 http://msg.wikidoc.info/index.php/Configutil_Reference 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 (issue the `msconfig edit mappings` command):

```
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` 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, 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=SUCCESES 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 enqueueing channel is `tcp_limited`.

See [MTA Text Databases](#) for more information on the Forward Text database.

Controlling Delivery Status Notification Messages

Starting with **Messaging Server 7 Update 5**, the MTA looks in both `serverroot/config/locale/C/` and `serverroot/lib/locale/C/` for DSN and MDN templates. As a result, the `configure` tool (used for initial Messaging Server configuration) does not create a `locale` subdirectory in the `config` directory. You can still customize DSN and MDN templates by creating the `locale` directory in the `config` directory and copying the templates that you want to edit.

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/lib/locale/C` directory (specified by the `mta.langdir` option, but if the `config/locale` directory is present, it takes precedence. The file names 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. Do not change the `lib/locale/C` files, as those are overwritten on any patch or upgrade. Copy files to the `config/locale/C` directory or `config/local/langcode` directory and edit them there. 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](#)
- [To Customize and Localize Delivery Status Notification Messages](#)
- [Internationalization of Generated Notices](#)
- [Additional Status Notification Message Features](#)

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-ascii
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 by the 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 <code>return_delayed.txt</code> and <code>return_timedout.txt</code> 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. Edit the `NOTIFICATION_LANGUAGE` table as follows:

```

msconfig
msconfig> unset -group mapping:NOTIFICATION_LANGUAGE
msconfig# edit mapping:NOTIFICATION_LANGUAGE

<Import the factory-supplied NOTIFICATION_LANGUAGE mapping from the
/opt/sun/comms/messaging/lib/mappings.locale file into your editor and
modify as desired.>

msconfig# write

```

Note

This procedure also works for the other mapping tables in the `configlib.xml` file, including the following:

```

mapping:LDAP_USERS_LANGUAGE
mapping:DOMAIN_DC
mapping:LANGUAGE_LOCALES
mapping:DISPOSITION_LANGUAGE
mapping:LDAP_USERS2_LANGUAGE

```

Except for `DISPOSITION_LANGUAGE`, these are all subsidiary mapping tables.

The `NOTIFICATION_LANGUAGE` mapping table 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

 *|*|en*|*|*           $IIMTA_TABLE:locale/C/,IMTA_LIB:locale/C/
 *|*|de*|*|*
 $IIMTA_TABLE:locale/de/,IMTA_LIB:locale/de/
 *|*|es*|*|*
 $IIMTA_TABLE:locale/es/,IMTA_LIB:locale/es/
 *|*|fr*|*|*
 $IIMTA_TABLE:locale/fr/,IMTA_LIB:locale/fr/
 *|*|ja*|*|*
 $IIMTA_TABLE:locale/ja/,IMTA_LIB:locale/ja/
 *|*|zh-TW*|*|* $IIMTA_TABLE:locale/zh_TW/,IMTA_LIB:locale/zh_TW/
 *|*|zh*|*|*
 $IIMTA_TABLE:locale/zh/,IMTA_LIB:locale/zh/
 *|*|ko*|*|*
 $IIMTA_TABLE:locale/ko/,IMTA_LIB:locale/ko/
 ! If a first language choice wasn't ok, check for additional language
 choices.
 *|*|$_*,$T**|*|*           $R$0|$1|$4|$5|$6
 !   For one's "own" users -- those users in the LDAP directory -- check
 if the
 ! user (or failing that their domain) has a preferredLanguage set that
 ! corresponds to a supported locale.
 *|*|*|*|*|*           $C$|LDAP_USERS_LANGUAGE;$3@$4|$E

```

In Unified Configuration, the `mappings.locale` file has been moved to the `configlib.xml` file (

SERVERROOT/lib/config.xml file). To disable notification language mapping, run the following commands:

```
msconfig> unset -group mapping:NOTIFICATION_LANGUAGE  
msconfig> unset -group mapping:DISPOSITION_LANGUAGE
```

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 described 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 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` option to 24. Return job scheduling is controlled by the `schedule.task:return_job.cron` option. 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, 1, 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: `posttheadonly`, `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
---------	-------------

<i>Returned Message Content</i>	<i>Specifies Addresses on Notifications</i>
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.
<i>Returned Messages</i>	<i>How to handle failure notices for returned messages.</i>
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.
<i>Warning Messages</i>	<i>How to handle warning messages.</i>
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.
<i>Returned Message Content</i>	<i>Specifies whether to send entire message or just headers to the postmaster.</i>
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.
<i>Returned Message Content</i>	<i>Specifies Addresses on Notifications</i>
includefinal	Include final form of address in delivery notifications (recipient address).

<code>returnenvelope</code>	Control use of blank envelope return addresses. The <code>returnenvelope</code> 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 <code>mailfromdnsverify</code> keyword.
<code>suppressfinal</code>	Suppress the final address form from notification messages, if an original address form is present, from notification messages.
<code>useintermediate</code>	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.
<i>Returned Message Content</i>	<i>Specifies Addresses on Notifications</i>
<code>aliaspostmaster</code>	Messages addressed to the user name <code>postmaster</code> at the official channel name is redirected to <code>postmaster@local-host</code> , where <code>local-host</code> is the local host name (the name on the local channel).
<code>returnaddress</code>	Specifies the return address for the local postmaster.
<code>noreturnaddress</code>	Use <code>RETURN_ADDRESS</code> option value as postmaster address name.
<code>returnpersonal</code>	Set the personal name for the local postmaster.
<code>noreturnpersonal</code>	Use <code>RETURN_PERSONAL</code> 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.

Topic in this section:

- [To Customize and Localize Message Disposition Notification 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. (Unlike most of the MTA configuration, the `DISPOSITION_LANGUAGE` mapping is contained in the `configlib.xml` file.) 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.txt disposition_denied.txt
disposition_prefix.txt disposition_dispatched.txt
disposition_processed.txt disposition_displayed.txt
disposition_suffix.txt disposition_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:

process_substitutions Options

Bit	Value	Description
0	1	Enables substitutions in <code>mgrpDisallowedBroadcaster</code> if set
1	2	Enables substitutions in <code>mgrpAllowedBroadcaster</code> if set
2	4	Enables substitutions in <code>mgrpModerator</code> if set
3	8	Enables substitutions in <code>mgrpDeliverTo</code> if set
4	16	Enables substitutions in <code>memberURL</code>

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?(&!(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_substitutions` setting to 2, and then you can set the following entries:

```
mgrpAllowedBroadcaster:  
ldap:///o=Sesta,c=US??sub?(&!(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 3. Administering Event Notification Service in Messaging Server for Unified Configuration

Administering Event Notification Service in Messaging Server for Unified Configuration

This information describes what you need to do to enable the Event Notification Service Publisher (ENS Publisher) and administer Event Notification Service (ENS) in Unified Configuration. For prior versions of Messaging Server, see the "Administering Event Notification Service in Messaging Server" chapter in *Messaging Server System Administrator's Guide*.

Topics:

- [ENS Publisher in Messaging Server](#)
- [Configuring the ENS Publisher in Unified Configuration](#)
- [Administering Event Notification Service](#)

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 the ENS publisher are bundled with Messaging Server.

Configuring the ENS Publisher in Unified Configuration

In Messaging Server 7 Update 5, ENS has the following default behavior:

- ENS is enabled by default. The initial configuration sets the `ens.enable` option to 1. If you upgraded to Messaging Server 7 Update 5, verify this setting.
- No configuration is required to load the ENS publisher because the `ms-internal` instance is automatically loaded and configured. Therefore, you do not need to create a separate `ms-internal` instance.
- If you want to configure options for the pre-loaded `ms-internal` default instance (as described in the "Administering Event Notification Service in Messaging Server" chapter in *Messaging Server System Administrator's Guide*), set them with the `ms-internal` instance name. For example, `notifytarget:ms-internal.settings`.
- IMAP IDLE now only works using ENS, because the ability to use IMAP IDLE with JMQ has been removed.
- The `notifytarget:ms-internal.enshost` defaults to `base.listenaddr` if it is not set.

Administering Event Notification Service

Administering ENS consists of starting and stopping the service, and changing the options to control the behavior of the ENS publisher.

Topics in this section:

- [Starting and Stopping ENS](#)
- [Event Notification Service Configuration Options](#)

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 Options

The `notifytarget:target.*` options control the behavior of the publisher. Use the `msconfig set` command to set these options. For a list of options, see http://msg.wikidoc.info/index.php/Configutil_Reference.

- To enable ENS, make sure that the `ens.enable` option is set to 1, for example:

```
/opt/sun/comms/messaging64/bin/msconfig set ens.enable 1
/opt/sun/comms/messaging64/bin/msconfig show ens.enable
role.ens.enable = 1
```

If you had set any of the `local.store.notifyplugin.*` parameters to a non-default value in a release prior to Messaging Server 7 Update 4, you have to set them again under the new name, `notifytarget:target.option`, when you upgrade to Messaging Server 7 Update 5 and use Unified Configuration.

Chapter 4. BURL Support for SMTP SUBMIT in Unified Configuration

BURL Support for SMTP SUBMIT in Unified Configuration

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, for example, by running `msconfig edit mappings` and adding the appropriate BURL mapping flags. 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:me
```

(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 `$:|` test; note that the `MAPPING_PARANOIA` option, if set, will cause any vertical bar within the client's original URL to be replaced in the probe by the specified alternate character (with the vertical bar input flag still being set). The mapping must set `$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 <i>url-types</i>	For the initial (EHLO) probe, enable advertising BURL support, optionally providing via the string argument <i>url-types</i> a space-separated list of the URL types to advertise.
\$N <i>string</i>	Ⓜ For the later (BURL command) probes, reject access. If the optional <i>string</i> is supplied, use it as the (entire) SMTP rejection, including SMTP error code and extended code as well as text. If no such <i>string</i> 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 <i>new-url</i>	Ⓜ For the later (BURL command) probes, if \$Y was also specified (allowing access), then use the specified <i>new-url</i> 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 <i>mailHost</i> attribute of the currently authenticated user. The BURL command probe itself will be considered to fail if the currently authenticated user has no <i>mailHost</i> 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 resembles the following (view by running `msconfig edit mappings`):

```
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*:* $:ASM$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*:* $:ASM$Y
! Allow BURL commands, connecting to (hosted domain) user's own mailHost
*|*@*|imap://*;URLAUTH=submit+$1%40$2*:* $:ASM$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 the following (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
```

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` options are used to accomplish this. `IMAP_USERNAME` specifies

the submit user and defaults to the setting of the `imap.submituser` option if not specified. `IMAP_PASSWORD` specifies the password which of course must 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](#).

Options for BURL

Option	Description
MAPPING_PARANOIA	<p>(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.</p>
IMAP_PASSWORD	<p>(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.</p>
IMAP_USERNAME	<p>(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 imap.submituser option. This feature was introduced in Message Server 7.</p>

Chapter 5. Configuring and Administering Multiplexor Services in Unified Configuration

Configuring and Administering Multiplexor Services in Unified Configuration

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 in Unified Configuration Overview](#)
- [Multiplexor Services](#)
- [About Messaging Multiplexor](#)
- [Setting Up the Messaging Multiplexor](#)
- [Configuring MMP with SSL](#)
- [MMP Tasks](#)

Multiplexor Services in Unified Configuration Overview

Starting with Messaging Server 7 Update 5, the MMP configuration is stored in the Unified Configuration. The following MMP configuration files are no longer used in Unified Configuration:

Legacy MMP Configuration Files

File Type	Legacy File Names
POP SSL MMP Encryption File	PopProxyAService.cfg
POP Services Configuration Template	PopProxyAService-def.cfg
IMAP SSL MMP Encryption File	ImapProxyAService.cfg
IMAP Services Configuration Template	ImapProxyAService-def.cfg
Service Starting Configuration File	AService.cfg
Service Starting Configuration Template	AService-def.cfg
SMTP SSL MMP Encryption File	SmtproxyAService.cfg
SMTP Services MMP Configuration Template	SmtproxyAService-def.cfg

In Unified Configuration, you enable and modify the MMP configuration by running the `msconfig` command to set the appropriate MMP options. The `ServiceList` and `SSLports` parameters are gone in Unified Configuration. You now use the `imapproxy`, `popproxy`, and `smtproxy` configuration groups, and the `tcp_listen` option. Use the following commands to view the initial MMP configuration settings starting with Messaging Server 7 Update 5:

```

msconfig
msconfig> show mmp*
role.mmp.enable = 0
msconfig> show imapproxy*
role.imapproxy.connlimits = :20
role.imapproxy.tcp_listen:imapproxy1.tcp_ports = 143
msconfig> show popproxy*
role.popproxy.connlimits = :20
role.popproxy.tcp_listen:popproxy1.tcp_ports = 110
msconfig> show submitproxy*
role.submitproxy.connlimits = :20

```

In addition, the `ssl_ports` option works the like `tcp_ports` option but enables SSL services (thus fixing the problem in legacy configuration, where an SSL proxy service had to be listed in both `ServiceList` and `SSLPorts` parameters).

The following examples commands show how to update the MMP configuration:

- To enable MMP:
`msconfig set mmp.enable 1`
- To change an IMAP proxy option:
`msconfig set imapproxy.optionvalue`
- To change a POP proxy option:
`msconfig set popproxy.optionvalue`
- To change an SMTP proxy option:
`msconfig set smtpproxy.optionvalue`
- To set a certmap default option:
`msconfig set base.certmap:default.optionvalue`

See [MMP Reference](#), or option descriptions in the `msconfig` online help, for more information.

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.

In Unified Configuration, MMP is no longer managed separately from Messaging Server. Note that 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 host 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

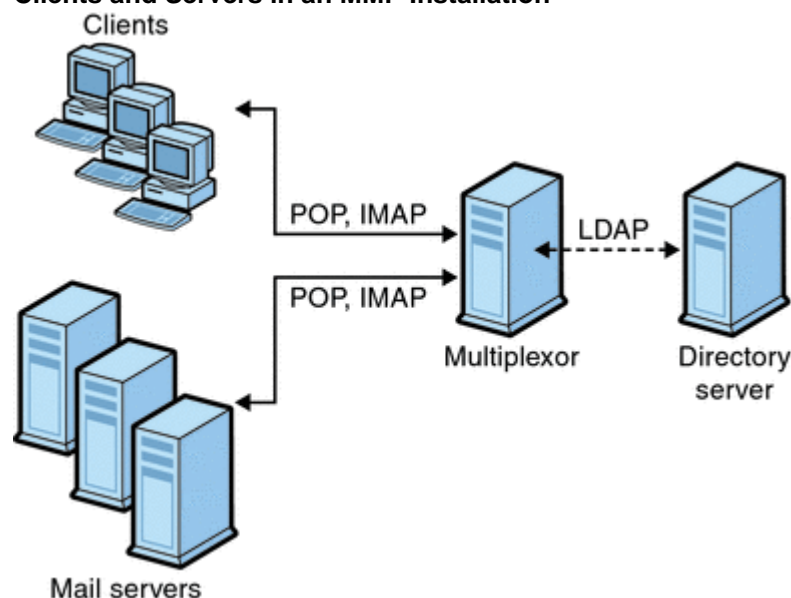
This section consists of the following subsections:

- [How the Messaging Multiplexor Works](#)
- [Encryption \(SSL\) Option](#)
- [Certificate-Based Client Authentication](#)
- [To Enable Certificate-based Authentication for Your IMAP or POP Service](#)
- [User Pre-Authentication](#)
- [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). [How the Messaging Multiplexor Works](#) shows how servers and clients relate to each other in an MMP installation. 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.

In legacy configuration, to enable SSL encryption for your IMAP, POP, and SMTP services, you would uncomment the appropriate SSL settings from the `.cfg` files. In Unified Configuration, you use the `msconfig` command to set the appropriate options. You must also set 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. To enable SSL, you must install an SSL server certificate. Then, you should use the `msconfig` command to set the SSL parameters. For a list of the SSL parameters, see the `ssl*` options in the `msconfig` online help.

Certificate-Based Client Authentication

In Unified Configuration, the certificate mapping file (`certmap.conf`), which matches a client's certificate to the correct user in the Users/Groups Directory Server, is no longer used. Instead, you use the following `msconfig` command to set the appropriate options:

```
msconfig set base.certmap:default.<option> <value>
```

The `default` can be replaced with the certificate `issuerDN` to have configuration specific to that certificate. That replaces the other groups in the `certmap.conf` file.

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.

In Unified Configuration, the MMP supports the `dncomps` and `filtercomps` options. The values of these two options has the form `fromattr=toattr`. 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:

```
msconfig> set base.certmap:default.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 default `certmap`

option in your configuration.
For example:

```
msconfig set base.certmap:default.dncomps ""
```

4. Install at least one trusted CA certificate, as described in [To Install Certificates of Trusted CAs](#).

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 0xhex) 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 hexadecimal 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. The `hosteddomains` option defaults to 1 (enabled).

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.

To configure a virtual domain option, use the following command:

```
msconfig set vdomain:<IP-address>.<option> <value>
```

For example, to set the default domain, use the following command:

```
msconfig set vdomain:192.0.2.0.defaultdomain example.com
```

When set, virtual domain configuration parameter values override global configuration option values.

You can specify the following configuration options for a virtual domain:

```
authcachettl
authenticationldapattributes
authservice
authservicettl
binddn
bindpass
clientlookup
crams
debugkeys
defaultdomain
domainsearchformat
ehlokeywords
failovertimeout
hosteddomains
ldapcachesize
ldapcachettl
mailhostattrs
popbeforesmtpkludgechannel
preauth
replayformat
restrictplainpasswords
searchformat
smtpproxypassword
smtprelays
ssladjustciphersuites
sslnicknames
storeadmin
storeadminpass
tcpaccess
tcpaccessattr
virtualdomaindelim
```

For more information on these configuration options, see the `msconfig` online help or the [MMP Reference](#).



Tip

To view the reference information for these configuration options, use the following URL:

```
http://msg.wikidoc.info/index.php/MMP_Reference#<optionname>
```

For example, to see the description for the `tcpaccess` option, use the following URL:

```
http://msg.wikidoc.info/index.php/MMP_Reference#tcpaccess
```

About SMTP Proxy

The MMP includes an SMTP submission 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 using the MMP SMTP proxy in *Unified Communications Suite Deployment Planning Guide*. 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:

- [Before You Configure MMP](#)
- [Multiplexor Configuration](#)
- [To Configure the MMP](#)
- [Multiplexor Configuration Parameters](#)
- [Starting the Multiplexor](#)
- [Modifying an Existing MMP](#)

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 do not conflict with one another.

2. On the machine where 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 preparing Directory Server for Messaging Server configuration in *Messaging Server 8.0 Installation and Configuration Guide*.
4. If the MMP is upgraded before the back-end servers, set the `capability` option to match the response to the `capability` command from the older back end. See the [capability](#) reference for more information.

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*

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. See the topic on creating the initial Messaging Server runtime configuration in *Messaging Server 8.0 Installation and Configuration Guide*.

Multiplexor Configuration Parameters

You control how the MMP operates by specifying various configuration options in the Unified Configuration. See [MMP Reference](#) 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/bin* directory.

MMP Commands

Option	Description
<code>start-msg mmp</code>	Starts the MMP (only if the MMP is enabled and one is not already running).
<code>stop-msg mmp</code>	Stops the most recently started MMP.
<code>refresh mmp</code>	Causes an MMP that is already running to refresh its configuration without disrupting any active connections.

Modifying an Existing MMP

1. To modify an existing instance of the MMP, use the `msconfig` command to edit the configuration as necessary.
2. The run either `refresh mmp` or `stop-msg mmp; start-msg mmp`. Use the former only if you changed "refreshable" options and the latter if you changed any "non-refreshable" options.

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. Generate and install the certificate by using the `certutil` command. See *Unified Communications Suite Certificate Authentication Guide* for details.
2. Set the password used for the certificate file.
For example:

```
msconfig
msconfig> set "sectoken:Internal (Software) Token.tokenpass"
newpassword
msconfig> write
```

The default setting for this password was provided during initial configuration, but it might be different.

It must match the password that was used when the certificate db was created by running the `certutil -N` command.

3. Set either `sslenable` on the relevant proxy (for STARTTLS) and/or set the `ssl_ports` on a `tcp_listen` for the appropriate proxy.
In general, the default settings cover the remainder of the configuration and you do not need to be changed.

1. Start the MMP:

```
<msg-svr-base>/bin/start-msg
```

2. If you do not want to use SSL between the MMP and the back-end server, then set the `sslbacksideport` option to 0 for `imapproxy` and `popproxy` as appropriate.

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 [Specifying Administrator Access to the Message Store in Unified Configuration](#).
4. Create a `certmap.conf` file for the MMP. For example:

```
msconfig> set base.certmap:default.dncomps ""
msconfig# set base.certmap: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. Use the `msconfig` command to update the configuration with the following options:
 - a. Set `storeadmin` and `storeadminpass` to values from Step 3.
 - b. Set `usergroupdn` to the root of your Users and Groups tree.
6. If you want client certificates with POP3, repeat Step 5 for the `popproxy` group.
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

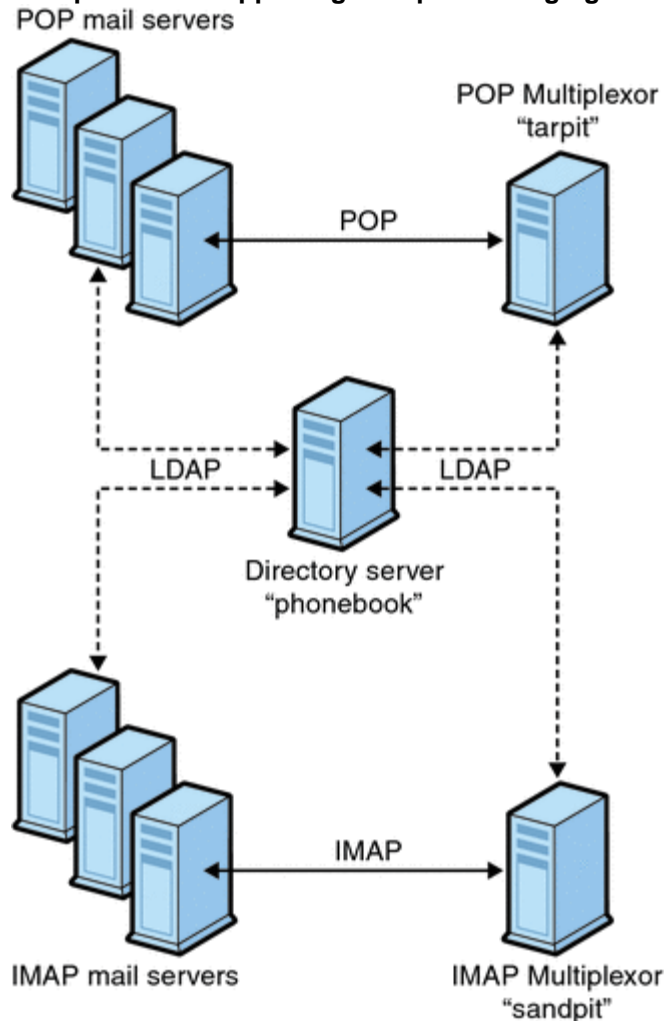
All your users have to perform this step if you want to use client certificates everywhere.

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 `imapproxy` entry from the MMP configuration. Likewise, you can restrict client access to IMAP services alone by removing the `popproxy` entry from the MMP configuration). 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



MMP Tasks

This section describes the following miscellaneous MMP configuration tasks:

- 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 `tcpaccessattr` option.

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 the `ugldaphost` option. For example:

```
msconfig set ugldaphost "ldap1.example.com ldap2.example.com"
```



Note

Make sure there is a space between the host names in the preceding configuration, and because of that space, to enclose the hosts in quotation marks.

Chapter 6. Configuring General Messaging Capabilities in Unified Configuration

Configuring General Messaging Capabilities in Unified Configuration

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 elsewhere.

Topics:

- [Modifying Your Passwords](#)
- [Managing Mail Users, Mailing Lists and Domains](#)
- [Starting and Stopping Services](#)
- [Automatic Restart of Failed or Unresponsive Services](#)
- [Scheduling Automatic Tasks](#)
- [Configuring a Greeting Message](#)
- [Setting a User-Preferred Language](#)
- [Encryption Settings](#)
- [Setting a Failover LDAP Server](#)

Modifying 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 password options that are set up during initial runtime configuration. Use the `msconfig` command to make changes to the Messaging Server configuration, or `ldapmodify` to update information stored in Directory Server.

Passwords Set in Messaging Server Initial Runtime Configuration

Option	Description
<code>base.ugldapbindcred</code>	Password for the Messaging Server LDAP user/group access account (<code>base.ugldapbinddn</code>). Use <code>msconfig</code> to change.
<code>base.proxyadminpass</code>	Password for the Proxy Administrator account (<code>base.proxyadmin</code>), which is used to provide proxy authentication access to end-user mailboxes. Use <code>msconfig</code> to change.
<code>http.smtpauthpassword</code>	Password used when <code>mshttpd</code> submits mail to the MTA. Set by initial configuration to the same password as <code>base.proxyadminpass</code> . Use <code>msconfig</code> to change.
SSL passwords for key files	Passwords that are stored in the <code>xpass.xml</code> file. Use the <code>msconfig set -prompt "sectoken:Internal (Software) Token"</code> command to change. This command causes <code>msconfig</code> to prompt for the password without an echo.
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 <code>base.proxyadminpass</code> and <code>http.smtpauthpassword</code> .
Messaging End User Administrator	This is the LDAP user for this specific host. The <code>base.ugldapbindcred</code> entry and the "Messaging End User Administrator" actually refer to the same password, which is set both in the option and in the <code>userPassword</code> 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 `proxyadminpass` option to change the password of the Proxy Administrator account. You should not set passwords from the command line, so this example shows using `msconfig` in interactive mode.

```
msconfig
msconfig> set -prompt proxyadminpass
Password:
Verify:
msconfig# write
msconfig> exit
```

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.

Topics in this section:

- [Overview of Messaging Server and LDAP](#)
- [To Remove a User from Messaging Server by Using Delegated Administrator](#)
- [To Remove a Domain from Messaging Server using Delegated Administrator](#)

Overview of Messaging Server and LDAP

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 by 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 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 Server 7, the command is `davadmin account delete`. (See the topic on removing Calendar users in *Calendar Server System Administrator's Guide* for more information.) For Calendar Server 6, 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 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 Server 7, the command is `davadmin account delete`. (See the topic on removing Calendar users in *Calendar Server System Administrator's Guide* for more information.) For Calendar Server 6, the program is `csclean`.
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.

Topics in this section:

- [To Start and Stop Messaging Server Services in an HA Environment](#)
- [To Start and Stop Messaging Server Services in a non-HA Environment](#)
- [To Start Up, Shut Down, or View the Status of Messaging Services](#)
- [Starting and Stopping a Messaging Server Running in MTA-only Mode](#)

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`.

The finest granularity in Oracle Solaris Cluster (formerly known as Sun Cluster) is that of an individual resource. Because Messaging Server is known to Solaris Cluster as a resource, the Solaris 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	<code>scswitch -e -j resource</code>	<code>sscswitch -Z -g resource_group</code>
Restart	<code>scswitch -n -j resource</code> <code>scswitch -e -j resource</code>	<code>scswitch -R -g resource_group</code>
Stop	<code>scswitch -n -j resource</code>	<code>scswitch -F -g resource_group</code>

Start, Stop, Restart in a Sun Cluster 3.2 Environment

Action	Individual Resource	Entire Resource Group
Start	<code>clrs online resource</code>	<code>clrg online resource_group</code>
Restart	<code>clrs disable resource</code> <code>clrs enable resource</code>	<code>clrg restart resource_group</code>
Stop	<code>clrs offline resource</code>	<code>clrg disable resource_group</code>

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

Action	Individual Resource	Entire Resource Group
Start	<code>hares -online resource -sys_system_</code>	<code>hagrp -online group -sys_system_</code>
Restart	<code>hares -offline resource -sys system</code> <code>hares -online resource -sys system</code>	<code>hagrp -offline group -sys system</code> <code>hagrp -online group -sys system</code>
Stop	<code>hares -offline resource -sys_system_</code>	<code>hagrp -offline group -sys_system_</code>

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 by using the `start-msg` and `stop-msg` commands.



Note

You must first enable services 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, it is generally safer to 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 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 the documentation on the `start-msg` and `stop-msg` commands in *Messaging Server System Administrator's Guide*.

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
```

The complete list of scopes with an "enable" option is as follows. Some of these, such as `notifytarget`, `service`, and `task` are named scopes. For more information, see [scope syntax](#).

```
autorestart
notifytarget
ens
folderquota
imap
indexer
messagetype
msghash
dbreplicate
mta
metermaid
mmp
pab
pop
purge
relinker
schedule
service
smime
sms_gateway
snmp
store
task
typequota
watcher
http
```

Set both `imap.enable` and `imap.enablesslport` to 0 to disable IMAP. The same goes for POP and HTTP. See [Messaging Server 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, or pop 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 commands:

```
msconfig set store.enable 0
msconfig set http.enable 0
```

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

```
# 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

Topics in this section:

- [Overview of Messaging Server Monitoring Processes](#)
- [Automatic Restart in High Availability Deployments](#)

Overview of Messaging Server Monitoring Processes

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 (<code>stored</code>), <code>imsched</code> , MMP. (LMTP/SMTP servers are monitored by the dispatcher and LMTP/SMTP clients are monitored by the <code>job_controller</code> .) The <code>watcher</code> 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 <code>ims_master</code> , <code>lmtplib_server</code> , and store utilities.	IMAP, POP, HTTP, cert, job controller, message store (<code>stored</code>), <code>imsched</code> , ENS, LMTP, SMTP

Enabling the watcher (`watcher.enable 1`, default) monitors process failures and unresponsive services and logs error messages to the `default` log file indicating specific failures. To enable automatic server restart, use `msconfig` to set `base.autorestart.enable 1`. By default, this parameter is set to `no (0)`.

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 `base.autorestart`). 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 `base.autorestart.enable` is enabled, 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 the `watcher.port` option.

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. The following is an example log:

```
watcher process 13425 started at Mon June 4 11:23:54 2012

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 configuration parameters to be set for automatic restart in high availability deployments:

HA Automatic Restart Parameters

Parameter	Description/HA Value
<code>watcher.enable</code>	Enable watcher on <code>start-msg</code> startup. Default is enabled (1).
<code>base.autorestart.enable</code>	Enable automatic restart of failed or frozen (unresponsive) servers including IMAP, POP, HTTP, job controller, dispatcher, and MMP servers. Default is enabled).
<code>base.autorestart.timeout</code>	Failure retry time-out. 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 shutdown and a failover to the other system occurs. The value (set in seconds) should be set to a period value longer than the <code>msprobe</code> interval. (See <code>schedule.task</code> : in the following section). Default is 600.
<code>schedule.task:msprobe.crontab</code>	<code>msprobe</code> runs <code>schedule</code> . A crontab style schedule string. Default is <code>5,15,25,35,45,55 * * * * lib/msprobe</code> . To disable, run the following command: <code>msconfig set schedule.task:msprobe.enable 0</code>

Scheduling Automatic Tasks

Topics in this section:

- [Overview of Scheduling Automatic Tasks](#)
- [Scheduler Examples](#)
- [Pre-defined Automatic Tasks](#)

Overview of Scheduling 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 `schedule.task` 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 option requires a command and a schedule on which to execute the command. The format is as follows:

```
schedule.task:<taskname>.crontab = <schedule>
```

where:

- *taskname* is the name of the command to run, for example, `expire`, `msprobe`, and so on.
- *schedule* is a non-empty string with the following 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 are required if specified. For example, setting the 17th day of the month and Tuesday only runs the command on the 17th day of a month when it is Tuesday.

If you modify scheduler, either restart the scheduler with the command `stop-msg sched` and `start-msg sched`, or refresh the scheduler by running `refresh sched`.

- To disable a scheduled task:

```
msconfig set schedule.task:<taskname>.enable = 0
refresh sched
```

Scheduler Examples

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

```
msconfig set schedule.task:expire.crontab "30 0,8,16 * * * bin/imexpire"
```

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

```
msconfig set schedule.task:msbackup.crontab "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:

```
schedule.task:expire.crontab = 0 23 * * * bin/imexpire
schedule.task:snapshot.crontab = 0 2 * * * bin/imdbverify -s -m
schedule.task:snapshotverify.crontab =
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
* * * * bin/imdbverify -m
```

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

```
schedule.task:purge.crontab = 0 0,4,8,12,16,20 * * * bin/imsimta purge
schedule.task:return_job.crontab = 30 0 * * * lib/return_job
```

The following automatic task is set and enabled for the message store:

```
schedule.task:msprobe.crontab = 5,15,25,35,45,55 * * * * lib/msprobe
```

Configuring a Greeting Message

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

Topics in this section:

- [To Create a New User Greeting](#)
- [To Set a Per-Domain Greeting Message](#)

To Create a New User Greeting

- To create a new-user greeting:

```
msconfig set base.welcomemsg <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:

```
msconfig set base.welcomemsg '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.) Alternatively, you can do this within the `msconfig` prompt so that you do not need to the \$. Simply run `msconfig`, then issue the `set option value` command.

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 `base.welcomemsg` 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 the Directory Server Reference.

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

Greeting Message Theory of Operations

Greeting messages can be set by both the LDAP attribute `mailDomainWelcomeMessage`, the `base.welcomemsg` option, and the `message_language:langcode.welcomemsg` option. The `base.welcomemsg` option is the default, the `message_language:langcode.welcomemsg` option is language-code specific. The order in which a message is 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
base.welcomemsg;lang-"$user_prefLang"
base.welcomemsg;lang-"$domain_prefLang"
base.welcomemsg;lang-"$gen.sitelanguage"
base.welcomemsg
```

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 `base.welcomemsg` option, if specified. If a user has a preferred language (set with the `preferredlanguage` LDAP attribute) and `base.welcomemsg;lang-user_prefLang` is set, the user will receive that welcome message at the time of their first log in to the server. If `base.welcomemsg;lang-gen.sitelanguage` is set, and `preferredlanguage` is not set, but the site language is set (using `base.sitelanguage` parameter), user will receive that message. If no language tag parameter is set and a untagged `base.welcomemsg` 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
base.welcomemsg;lang-de
base.welcomemsg;lang-en
base.welcomemsg
```

According to the algorithm, the message sent to the user is `base.welcomemsg;lang-de`.

Setting a User-Preferred Language

Topics in this section:

- [Overview of Setting a User-Preferred Language](#)
- [To Set a Domain Preferred Language](#)
- [To Specify a Site Language](#)

Overview of Setting 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:

```
msconfig set base.sitelanguage <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 `base.ugldaphost` to the multiple replicated LDAP servers.
For example:

```
msconfig set base.ugldaphost "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
```

Chapter 7. Configuring IMAP IDLE in Unified Configuration

Configuring IMAP IDLE in Unified Configuration

To configure IMAP IDLE in Unified Configuration requires at least Messaging Server 7 Update 5. For prior versions of Messaging Server or when using legacy configuration, see the topic on configuring IMAP IDLE in *Messaging Server System Administrator's Guide*.

Topics:

- [Benefits of Using IMAP IDLE](#)
- [Configuring IMAP IDLE with ENS in Unified Configuration](#)

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 in Unified Configuration

In Messaging Server 7 Update 5, IMAP IDLE with ENS has the following default behavior:

- ENS is enabled by default. The initial configuration sets the `ens.enable` option to 1. If you upgraded to Messaging Server 7 Update 5, 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

Make sure ENS is enabled by setting the `ens.enable` option to 1:

```
msconfig set ens.enable 1
```

To Configure IMAP IDLE with ENS

1. Configure the `enpd` server to allow (or restrict) connections only from the hosts running the message stores by configuring the `ens.domainallowed` and `ens.domainnotallowed` options as necessary.

For example, the following command allows access to the local host only:

```
msconfig set ens.domainallowed enpd:127.0.0.1
```

The following command allows access to the local host and all IP addresses `192.168.0.*` except `192.168.0.17`:

```
msconfig set ens.domainallowed  
'"enpd:192.168.0.0/255.255.255.0,127.0.0.1 EXCEPT 192.168.0.17"'
```

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.

2. Stop, then restart Messaging Server.

```
cd /opt/sun/comms/messaging/sbin  
stop-msg  
start-msg
```

3. 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 (Oracle Communications Messaging  
Server 7u5-4.07 64bit (built Mar 21 2012)
```

To Disable IMAP IDLE

- To disable IMAP IDLE, set the `ens.enable` option to 0 (default is 1).
For example:

```
msconfig set ens.enable 0
```

Chapter 8. Configuring Messaging Server for High Availability in Unified Configuration

Configuring Messaging Server for High Availability in Unified Configuration

The following information pertains to Messaging Server 7.0.5.29.0. For additional information about configuring Messaging Server 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 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 starting in Messaging Server 7, which supersedes corresponding information in the *Messaging Server 6.3 Administration Guide*.

The remainder of this page contains the following topics:

- [New Recipe for Unified Configuration](#)
- [Supported Versions of High-Availability Software in Messaging Server 7](#)
- [New Installation Methods for Messaging Server 7](#)
- [New Installation Paths for Messaging Server Oracle Solaris Cluster HA Agent 7.0](#)
- [Installing Messaging Server Oracle Solaris Cluster HA Agent in Solaris Zones](#)
- [Using the `useconfig` Utility](#)

New Recipe for Unified Configuration

Wherever you see references to the `ha_ip_config` script, in Unified Configuration, use the `HAConfig.rcp` recipe. (Recipes are installed in the `msg-svr-base/lib/recipes` directory.) To run a recipe, use the `msconfig run recipe` command. For example, to run the `HAConfig.rcp` recipe, type:

```
/opt/sun/comms/messaging64/bin/msconfig run HAConfig.rcp
```

Respond to the prompts according. The recipe configures the logical IP and sets the following options:

```
base.listenaddr
job_controller.listenaddr
dispatcher.service:SMTP_SUBMIT.listenaddr
dispatcher.service:SMTP.listenaddr
http.smtphost
metermaid.listenaddr
metermaid_client.server_host
```

The recipe verifies that the watcher is enabled, and if not, enables it; and also enables autorestart if not

already enabled.

Supported Versions of High-Availability Software in Messaging Server 7

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

For the latest supported versions and platforms, see the Release Notes in *Messaging Server 8.0 Installation and Configuration Guide*.

New Installation Methods for Messaging Server 7

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 64-bit Messaging Server 7 Update 5 *msg-svr-base* path instead:

```
/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, 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.

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 in the global zone only:

```
./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.

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 9. Configuring Rewrite Rules in Unified Configuration

Configuring Rewrite Rules in Unified Configuration

This information describes how to configure rewrite rules. Make sure you have read [About MTA Services and Unified Configuration](#) before using this information.

Topics:

- [Before You Begin](#)
- [Editing Rewrite Rules in Unified Configuration](#)
- [Rewrite Rule Structure](#)
- [Rewrite Rule Patterns and Tags](#)
- [Rewrite Rule Templates](#)
- [How the MTA Applies Rewrite Rules to an Address](#)
- [Template Substitutions and Rewrite Rule Control Sequences](#)
- [Handling Large Numbers of Rewrite Rules](#)
- [Testing Rewrite Rules](#)
- [Rewrite Rules Example](#)

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, 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 [Recompiling the MTA Configuration](#).

Editing Rewrite Rules in Unified Configuration

To make rewrite rules changes:

- Use the `msconfig edit rewrites` command to bring up the rewrite rules in the editor specified by the `EDITOR` shell variable of the account that you are logged in as.
For example:

```

msconfig edit rewrites
$* $A$E$F$U%$H$V$H@&/IMTA_HOST/
&/IMTA_HOST/ $U%$D@&/IMTA_HOST/
&/IMTA_DEFAULTDOMAIN/ $U%$D@&/IMTA_HOST/
. $U%$H$, $H@tcp_local-daemon
[] $E$R${INTERNAL_IP, $L}$U%[$L]@tcp_intranet-daemon
.&/IMTA_DEFAULTDOMAIN/ $U%$H.&/IMTA_DEFAULTDOMAIN/@tcp_intranet-daemon
* $U%$&0.&/IMTA_DEFAULTDOMAIN/
reprocess $U%reprocess.&/IMTA_HOST/@reprocess-daemon
reprocess.&/IMTA_HOST/ $U%reprocess.&/IMTA_HOST/@reprocess-daemon
process $U%process.&/IMTA_HOST/@process-daemon
process.&/IMTA_HOST/ $U%process.&/IMTA_HOST/@process-daemon
defragment $U%defragment.&/IMTA_HOST/@defragment-daemon
defragment.&/IMTA_HOST/ $U%defragment.&/IMTA_HOST/@defragment-daemon
conversion $U%conversion.&/IMTA_HOST/@conversion-daemon
conversion.&/IMTA_HOST/ $U%conversion.&/IMTA_HOST/@conversion-daemon
bitbucket $U%bitbucket.&/IMTA_HOST/@bitbucket-daemon
bitbucket.&/IMTA_HOST/ $U%bitbucket.&/IMTA_HOST/@bitbucket-daemon
hold-daemon $U%$H@hold-daemon
~
~
~
~
~
~
"/var/tmp/manage.v_a474" 18 lines, 1226 characters

```

You can also add rewrite rules as regular option values by using the `set` command. For example, a separate queue for messages sent to a slowly responding domain could be accomplished with the following commands:

```

msconfig
msconfig> show rewrite.rule
role.rewrite.rule = $* $A$E$F$U$H$V$H@&/IMTA_HOST/
role.rewrite.rule = &/IMTA_HOST/ $U$D@&/IMTA_HOST/
role.rewrite.rule = &/IMTA_DEFAULTDOMAIN/ $U$D@&/IMTA_HOST/
role.rewrite.rule = . $U$H$, $H@tcp_local-daemon
role.rewrite.rule = [] $E$R$ {INTERNAL_IP, $L} $U$ [ ] @tcp_intranet-daemon
role.rewrite.rule = .&/IMTA_DEFAULTDOMAIN/
$U$H.&/IMTA_DEFAULTDOMAIN/@tcp_intranet-daemon
role.rewrite.rule = * $U$&0.&/IMTA_DEFAULTDOMAIN/
role.rewrite.rule = reprocess
$U$reprocess.&/IMTA_HOST/@reprocess-daemon
role.rewrite.rule = reprocess.&/IMTA_HOST/
$U$reprocess.&/IMTA_HOST/@reprocess-daemon
role.rewrite.rule = process $U$process.&/IMTA_HOST/@process-daemon
role.rewrite.rule = process.&/IMTA_HOST/
$U$process.&/IMTA_HOST/@process-daemon
role.rewrite.rule = defragment
$U$defragment.&/IMTA_HOST/@defragment-daemon
defragment.&/IMTA_HOST/ $U$defragment.&/IMTA_HOST/@defragment-daemon
conversion $U$conversion.&/IMTA_HOST/@conversion-daemon
conversion.&/IMTA_HOST/ $U$conversion.&/IMTA_HOST/@conversion-daemon
bitbucket $U$bitbucket.&/IMTA_HOST/@bitbucket-daemon
bitbucket.&/IMTA_HOST/$U$bitbucket.&/IMTA_HOST/@bitbucket-daemon
hold-daemon $U$H@hold-daemon
msconfig> set rewrite.rule slow.com $U$D@TCP-SLOW
msconfig# set rewrite.rule .slow.com $U$H$D@TCP-SLOW
msconfig# set channel:tcp_slow.official_host_name TCP-SLOW
msconfig# set channel:tcp_slow.smtp
msconfig# set channel:tcp_slow.mx
msconfig# set channel:tcp_slow.pool SMTP_POOL
msconfig# set mapping:ORIG_SEND_ACCESS.rule "tcp_local|*|tcp_slow|*"
"$N$D30|Relaying not allowed without prior authentication"
msconfig# diff
< role.channel:tcp_slow.mx (novalue)
< role.channel:tcp_slow.official_host_name = TCP-SLOW
< role.channel:tcp_slow.pool = SMTP_POOL
< role.channel:tcp_slow.smtp (novalue)
< role.mapping:ORIG_SEND_ACCESS.rule = tcp_local|*|tcp_slow|*
$N$D30|Relaying not allowed without prior authentication
< role.rewrite.rule = .slow.com $U$H$D@TCP-SLOW
< role.rewrite.rule = slow.com $U$D@TCP-SLOW
msconfig# write -remark="Add separate queue for slow domain slow.com"
msconfig>

```



Note

The `set rewrite.rule` command always adds rules at the end of the list, so order-specific rules must be added in the desired order. Also, it is not presently practical to remove rewrite rules with the `unset` command.

Rewrite Rule Structure

The following example configuration 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. In the legacy configuration, the rewrite rules appear in the upper half followed by the channel definitions in the lower half.

```
! An example configuration file.!\n! This is only an example of a configuration file. It serves\n! no useful purpose and should not be used in a real system.\n!\n! Part I: Rewrite rules\na $U@a-daemon\nb $U@b-daemon\nc $U%c@b-daemon\nd $U%d@a-daemon\n!\n! 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. When you view mappings by using the `msconfig edit rewrite` command, you see something that resembles the following:

```
<pattern> <template>\n<pattern> <template>\n<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> ]\n<UserTemplate>@<ChannelTag>[ <controls> ]\n<UserTemplate>%<DomainTemplate>[ <controls> ]\n<UserTemplate>@<DomainTemplate>@<ChannelTag>[ <controls> ]\n<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:

```
hosta.subnet.siroe.com
.subnet.siroe.com
.siroe.com
```

Based on the rewrite rule patterns, an address of `jd@hosta.subnet.siroe.com` matches the `hosta.subnet.siroe.com` rewrite rule pattern. An address of `jd@hostb.subnet.siroe.com` matches the `.subnet.siroe.com` rewrite rule pattern. And an address of `jd@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.

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 the following table, 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](#).

In a legacy configuration, the MTA includes rewrite rules for the top-level Internet domains from the `internet.rules` file (`msg-svr-base/config/internet.rules`). In a Unified Configuration setup (starting with Messaging Server 7 Update 5), the contents of the `internet.rules` file have been absorbed into the main configuration and the individual entries for top-level domains have been removed and replaced by the following single rewrite rule:

```
. $U%$H$, $H@TCP-DAEMON
```

This rewrite rule replaces the reference to the `internet.rules` file that appears in legacy configurations as the following:

```
<IMTA_TABLE: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, the configuration must contain 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

When additional Top Level Domains (TLDs) are defined by the IANA, perform the following based on whether or not you are using a Unified Configuration or a legacy configuration:

To Update TLDs in a Unified Configuration

1. Obtain a new copy of the `tlds.txt` file from the Internet Assigned Numbers Authority (IANA).
2. Put the `tlds.txt` file in the `msg-svr-base/config` directory.
3. Run the following command:

```
imsimta chbuild
```

To Update TLDs in a Legacy Configuration

1. Obtain a new copy of the `tlds.txt` file from the Internet Assigned Numbers Authority (IANA).
2. Put the `tlds.txt` file in the `msg-svr-base/config` directory.
3. Look at the `tlds.txt` file to see if the top-level domains are in your `msg-svr-base/config/internet.rules` file.
4. Append any new rules to the existing rules in the `internet.rules` file by using the following format:

```
! <New TLD Entry>
!
.<NEW-TLD-Entry> $U%$H$D@TCP-DAEMON
!
```

5. Run the following command:

```
imsimta chbuild
```

Because these are not "tagged" rewrite rules, the order that they appear in the `internet.rules` file is not significant.

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 later in this information).

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 `jd@com1` into the source-routed address `@siroe.com:jd@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. 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](#)
- [Step 2. Scan the Rewrite Rules](#)
- [Step 3. Rewrite Address According to Template](#)
- [Step 4. Finish the Rewrite Process](#)
- [Rewrite Rule Failure](#)
- [Syntax Checks After Rewrite](#)
- [Handling Domain Literals](#)

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` option 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` option.

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	No comment.
user%A%B	B	No comment.
user%%A%B	B	No comment.
A!user	A	"Bang-style" addressing; commonly used for UUCP.
A!user@B	B	No comment.
A!user%B@C	C	No comment.
A!user%B	B	<code>nobangoverpercent</code> option active; the default.
A!user%B	A	<code>bangoverpercent</code> option 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` options 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 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` option, 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 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:

- [Username and Subaddress Substitution, \\$U, \\$OU, \\$1U](#)
- [Host/Domain and IP Literal Substitutions, \\$D, \\$H, \\$nD, \\$nH, \\$L](#)

- Literal Character Substitutions, \$\$, \$%, @\$
- LDAP Query URL Substitutions, \$...[
- General Database Substitutions, \$(...)
- Apply Specified Mapping, \${...}
- Customer-supplied Routine Substitutions, \$[...]
- Single Field Substitutions, \$&, \$!, \$*, \$#
- Unique String Substitutions
- Source-Channel-Specific Rewrite Rules (\$M, \$N)
- Destination-Channel-Specific Rewrite Rules (\$C, \$Q)
- Direction-and-Location-Specific Rewrite Rules (\$B, \$E, \$F, \$R)
- Host-Location-Specific Rewrites (\$A, \$P, \$S, \$X)
- Changing the Current Tag Value, \$T
- Controlling Error Messages Associated with Rewriting (\$?)

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 `jd@siroe.com` was being rewritten by this template, the resulting output would be `jd@siroe.com`, the \$U substituting in the *username* portion, `jd`, 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.
\$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.
\$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 <i>n</i> th part of unmatched (or wildcarded) host, counting from left to right, starting from 0.
\$!n	The <i>n</i> th part of unmatched (or wildcarded) host, as counted from right to left, starting from 0.
\$*n	The <i>n</i> th part of matching pattern, counting from left to right, starting from 0.
\$#n	The <i>n</i> th part of matching pattern, counted from right to left, starting from 0.
\$nD	Portion of domain specification that matched, preserving from the <i>n</i> th leftmost part starting from 0
\$nH	Portion of host/domain specification that didn't match, preserving from the <i>n</i> th 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
\$Cchannel	Fail if sending to <i>channel</i>
\$E	Apply only to envelope addresses
\$F	Apply only to forward-directed (e.g., To\:) addresses
\$Mchannel	Apply only if <i>channel</i> is rewriting the address
\$Nchannel	Fail if <i>channel</i> is rewriting the address
\$P	Apply if host is to the right of a percent sign
\$Qchannel	Apply if sending to <i>channel</i>
\$R	Apply only to backwards-directed (e.g., From\:) addresses
\$S	Apply if host is from a source route
\$Tnewtag	Set the rewrite rule tag to newtag
\$Vhost	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
\$Zhost	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.
\$nT	Overrides the default ALIAS_MAGIC setting, where <i>n</i> 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 <i>errmsg</i> instead of the default error message. The error message must be in US ASCII.

<code>\$number?</code> <code>errmsg</code>	<p>If rewriting fails, return <i>errmsg</i> instead of the default error message, and set the SMTP extended error code to <i>a.b.c</i>:</p> <ul style="list-style-type: none"> • <i>a</i> is <i>number</i>/ 1000000 (the first digit) • <i>b</i> is (<i>number</i>/1000) remainder 1000 (the value of the digits 2 through 4) • <i>c</i> is <i>number</i> remainder 1000 (the value of the last three digits). <p>The following example sets the error code to 3.45.89: <code>\$3045089?the snark is a boojum</code></p>
---	--

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 `jdoue@host.siroe.com` matches the following rewrite rule:

```
host.siroe.com $U%$1D@tcp_local-daemon
```

The resulting address is `jdoue@siroe.com` with `tcp_local-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 with the `ugldaphost` and `ugldapport` options.

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
<code>\$\$</code>	Literal \$ character
<code>\$.</code>	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.
<code>\$~</code>	<i>account</i> Home directory of user account
<code>\$A</code>	Address
<code>\$D</code>	Domain name
<code>\$H</code>	Host name (first portion of fully qualified domain name)
<code>\$L</code>	Username minus any special leading characters such as <code>~</code> or <code>_</code>
<code>\$S</code>	Subaddress
<code>\$U</code>	Username

The MTA caches URL results from lookups done in rewrite rules and mappings. This URL result cache is controlled by the `URL_RESULT_CACHE_SIZE` (default 10000 entries) and `URL_RESULT_CACHE_TIMEOUT` (default 600 seconds) options.

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 `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 `jd@eng.siroe.com` 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 `jd@eng.siroe.com` (that is, `username = jd`, `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.



Note

If you are using a crdb, on-disk database (the `use_text_databases` option has not been set with the bit saying to use a "text" database for the general database), then `generaldb.db` is the specific file containing the general database, in the hard-coded location `msg-svr-base/data/db`.

If instead you are using a general "database" that is constructed by the MTA as an in-memory database built from an underlying text file, then the underlying text file is `IMTA_TABLE:general.txt` and the relevant bit (bit 0/value 1) of the `use_text_databases}]` option is set. In this case, no database file resides in the `msg-svr-base}{{/data/db` directory.

All the MTAs' regular crdb databases (alias database, reverse database, domain database, forward database, and general database) are expected to reside in the the `msg-svr-base /data/db` directory using hard-coded names. However, the general database, reverse database, and forward database can optionally be constructed in-memory by the MTA, as controlled by the `use_text_databases` option. In this case, instead of having a true crdb file located in the `msg-svr-basedata/db/` directory, there would be a text file in `IMTA_TABLE:.`

Apply Specified Mapping, \${...}

A substitution of the form `.SIROENET $(H) ${mapping,argument}` is used to find and apply a mapping from the MTA configuration. The `mapping` field specifies the name of the mapping table to use while `argument` specifies the string to pass to the mapping. The mapping must exist and must set the `$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 (that is, 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
<code>\$&n</code>	Substitute the <i>n</i> th element, <i>n</i> =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.
<code>\$!n</code>	Substitute the <i>n</i> th element, <i>n</i> =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.
<code>\$*n</code>	Substitute the <i>n</i> th element, <i>n</i> =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.
<code>\$#n</code>	Substitute the <i>n</i> th element, <i>n</i> =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 `jdoue@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 `jdoue@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 options `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 `rules` option 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, `$Mchannel` causes the rule to fail if `channel` is not currently doing the rewriting. `$Nchannel` 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 `rules` and `norules` options 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 `rules` option 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, `$Qchannel` causes the rule to fail if `channel` is not the destination. For another example, `$Cchannel` 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_local-daemon` channel.

This could be implemented with rules like the following (`localhost` is assumed to be the official name of the local host):

```
internet $$U@localhost$Tmtcp-force|
mtcp-force|. $U%$H@tcp_local-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_local-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 rewrite rules 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 in the `msg-svr-base{/data/db/` directory. 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.

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
* $U%$&0.cs.siroe.edu
*.cs $U%$&0.cs.siroe.edu
*.cs.siroe $U%$&0.cs.siroe.edu
*.cs.siroe.edu $U%$&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@cads.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 10. Integrating Spam and Virus Filtering Programs Into Oracle Communications Messaging Server in Unified Configuration

Integrating Spam and Virus Filtering Programs Into Oracle Communications Messaging Server in Unified Configuration

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](#)
- [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](#)

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 with MTA options, 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 options had names, such as `sourcebrightmail` or `Brightmail_config_file`. These 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, using MTA options, the spam filtering software library path (`spamfilterX_library`) and configuration file (`spamfilterX_config_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 options; an *X* is used as a filter identification number. For example, `sourcespamfilterXoptin` or `spamfilterX_config_file`. If the identifying number is omitted from the keyword or option name it defaults to 1.

The following MTA option settings specify Messaging Server to filter messages through both Symantec AntiVirus Scan Engine and SpamAssassin:

```
msconfig
msconfig> set spamfilter1_library=<Symantec_Library_File>
msconfig# spamfilter1_config_file=<Symantec_Config_File>
msconfig# spamfilter2_library=<SpamAssassin_Library_File>
msconfig# spamfilter2_config_file=<SpamAssassin_Config_File>
msconfig# write
```

When using other options 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_OPTINX` MTA options. Example:

```
msconfig> set ldap_optin1 SymantecAV
msconfig# set ldap_optin2 SpamAssassin
msconfig# write
```

**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_OPTINI1` 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_OPTINX MTA options. Example:

```
msconfig
msconfig> set ldap_domain_attr_optin1 SymantecAV
msconfig# set ldap_domain_attr_optin2 SpamAssassin
msconfig# write
```

**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 the `LDAP_DOMAIN_ATTR_OPTIN1` MTA option was set to `Brightmail`. 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 options described in [Spamfilter channel options](#). The following example demonstrates how to set up channel-level filtering.

1. Add a rewrite rule for all inbound SMTP servers that send messages to a backend message store host by running `msconfig edit rewrite` and adding, for example:

```
msg_store1.siroe.com $U@msg_store1.siroe.com
```

2. Add a channel corresponding to the rewrite rule with the `destinationspamfilterXoptin` option by running `msconfig edit channels` and adding, for example:


```
tcp_msg_store1 smtp subdirs 20 backoff "pt5m" "pt10" "pt30" \  
"pt1h" "pt2h" "pt4h" maxjobs 1 pool IMS_POOL \  
fileinto $U+$S@$D destinationspamfilterloptin spam  
msg_store1.siroe.com
```

Channel-level Filtering Examples

These examples assume a filtering program specified by the number 1. See [Spamfilter channel options](#) for the options 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 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 `destinationspamfilterXoptin` option. Example:

```
tcp_msg_store1 smtp subdirs 20 backoff "pt5m" "pt10" "pt30" "pt1h" \  
\  
"pt2h" "pt4h" maxjobs 1 pool IMS_POOL fileinto $U+$S@$D \  
destinationspamfilterloptin spam,virus  
msg_store1.siroe.com
```

Example 2. Filter for spam all incoming mail passing through your MTA with the `sourcespamfilterXoptin` option. (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 \  
maytlserver maysaslserver saslswitchchannel tcp_auth \  
sourcespamfilterloptin 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 \  
maytlserver maysaslserver saslswitchchannel tcp_auth \  
destinationspamfilterloptin 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 \  
maytllserver maysaslserver saslswitchchannel tcp_auth \  
sourcespamfilterloptin spam destinationspamfilterloptin 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 \  
maytllserver maysaslserver saslswitchchannel tcp_auth \  
destinationspamfilterloptin 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 \  
maytllserver maysaslserver saslswitchchannel tcp_auth \  
destinationspamfilterloptin spam,virus \  
sourcespamfilterloptin 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 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 `spamfilterX_verdict_n` and `spamfilterX_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

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 <code>syslog</code> . 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 <code>inetMailUser</code> objectclass. The attribute itself can take multiple values and is case-sensitive. For SpamAssassin, its value should be <code>spam</code> 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 <code>mailDomain</code> . Default: none

spamfilterX_null_optin	<p>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.</p> <p>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.)</p>
spamfilterX_null_action	<p>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.</p> <p>Default: data: ,discard;</p>
spamfilterX_string_action	<p>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.</p> <p>Default: data: ,require "fileinto"; fileinto "\$U; where \$U is the string that verdict returned.</p>
spamfilterX_verdict_n	<p>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:</p> <pre>spamfilter1_verdict_0=reject spamfilter1_action_0=data:,require "reject"; reject "Rejected by spam filter";</pre> <p>The default values for all the spamfilterX_verdict_n options and the corresponding action options are empty strings.</p> <p>Default: none</p>
spamfilterX_action_n	<p>See spamfilterX_verdict_n.</p> <p>Default: none</p>

spamfilterX_final	<p>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 <code>spamfilterX_final</code>. 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 <code>spamfilterX_final</code> 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 <code>spamfilterX_final</code> 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 <code>spamfilterX_final</code> 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.</p> <p>Default: 0</p>
spamfilterX_returnpath	<p>(New in Messaging Server 7 Update 1) If set to 1 this will cause a <code>Return-path:</code> header to be added to the message passed to the corresponding filter. 0 disables this addition.</p> <p>Default: 0</p>
optin_user_carryover	<p>Forwarding is a challenge for spam filter processing. Consider a user entry that specifies the <code>forward</code> 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 <code>OPTIN_USER_CARRYOVER</code> 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:</p> <p>bit 0 (value 1). Each LDAP user entry overrides any previously active user/domain optins unconditionally.</p> <p>bit 1 (value 2). If a user's domain has an optin attribute, it overrides any previous user/domain/alias optins that were active.</p> <p>bit 2 (value 4). If a user has an optin attribute, it overrides any previous user/domain/alias optins that were active.</p> <p>bit 3 (value 8). An optin specified by an <code>[optin]</code> non-positional parameter overrides any previous user/domain/alias optins that were active.</p> <p>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.)</p>

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

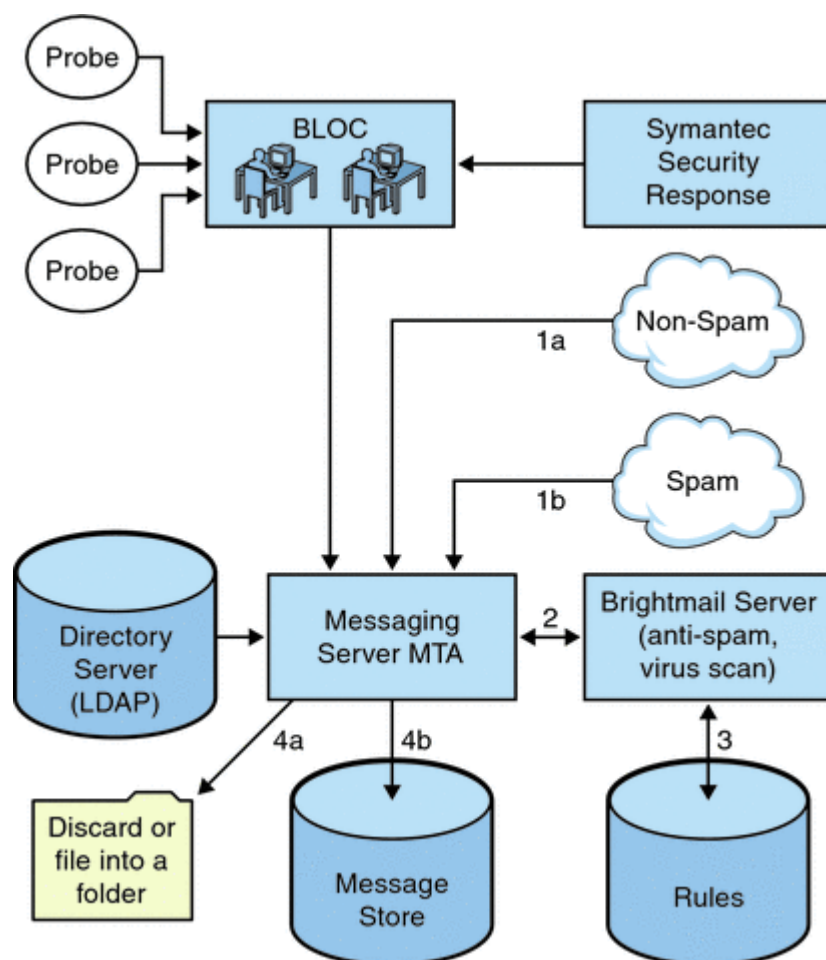
How Brightmail Works

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 (MTA channel configuration)
- Whether there is a channel default for the services opted in (MTA channel configuration)
- 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
<code>blSWPrecedence</code>	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 <code>virus-spam</code> the verdicts are separated by hyphens (-). This is the recommended setting when using Brightmail with Sun Java System Messaging Server.
<code>blSWClientDestinationDefault</code>	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 <code>inbox</code> as the value. There is no default.
<code>blSWLocalDomain</code>	<p>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.</p> <p>See below <code>blSWClientDestinationLocal</code> and <code>blSWClientDestinationForeign</code>. For example, you can specify</p> <pre>blSWLocalDomain=siroe.com</pre>

blSWClientDestinationLocal	<p>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 <code>verdict action</code>, For example,</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre>blSWClientDestinationLocal=spam spambox blSWClientDestinationLocal=virus </pre> </div> <p>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 <code>virus</code>. And if the verdict is <code>spam</code>, the above example files the message into the folder called <code>spambox</code>. 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 <code>blSWClientDestinationDefault</code> setting above.</p> <p>When using a separate Brightmail server or servers from the MTA, you can customize the actions taken by each MTA by using the <code>Brightmail_verdict_n</code>, <code>Brightmail_action_n</code>, <code>Brightmail_null_action</code>, and <code>Brightmail_string_action</code> MTA options to override the actions and verdicts returned by the Brightmail server. In this example, you can use different <code>Brightmail_null_action</code> on the MTA to override the Virus action (which would be to discard it) or to use <code>Brightmail_verdict_0=spambox</code>, and <code>Brightmail_action_0=data:,require "fileinto";fileinto "Junk";</code> to file into a folder called <code>Junk</code> instead of <code>spambox</code>.</p>
blSWClientDesintationForeign	<p>Same format and interpretation as <code>blSWClientDestinationLocal</code> above, except this applies to users in the domain which are NOT local.</p>
blSWUseClientOptin	<p>Always set this to TRUE when used with Sun Java System Messaging Server.</p>
blswcServerAddress	<p>Is of the form <code>ip:port[, ip:port, ...]</code> to specify one or more Brightmail server's IP address and port numbers</p>

Using SpamAssassin

This section consists of the following subsections:

- [SpamAssassin Overview](#)
- [SpamAssassin/Messaging Server Theory of Operations](#)
- [SpamAssassin Requirements and Usage Considerations](#)
- [Deploying SpamAssassin](#)
- [SpamAssassin Configuration Examples](#)
- [Testing SpamAssassin](#)
- [SpamAssassin Options](#)

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. Refer to [Spamfilter channel options 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 MTA options:

```
msconfig
msconfig> set spamfilter1_config_file
IMTA_TABLE:spamassassin.opt
msconfig# set spamfilter1_config_file
IMTA_TABLE:spamassassin.opt
msconfig# set spamfilter1_library IMTA_LIB:libspamass.so
msconfig# set spamfilter1_optional 1
msconfig# set spamfilter1_string_action=data:, require
["editheader","spamtest"]; \
spamadjust "$U"; addheader "X-Spam-Score: $U"
msconfig# write
```

- 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 mapping table `msconfig edit mappings` 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 (`msconfig edit channels`):

```

!
! conversion_peruser
conversion_peruser single sourcespamfilterloptin 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 (`msconfig edit mappings`):

```

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 MTA options:

```
msconfig
msconfig> set spamfilter1_config_file
/opt/sun/comms/messaging/spamassassin.opt
msconfig# set spamfilter1_config_file
/opt/sun/comms/messaging/spamassassin.opt
msconfig# set spamfilter1_library
/opt/sun/comms/messaging/lib/libspamass.so
msconfig# set spamfilter1_optional 1
msconfig# set spamfilter1_string_action=data:,require "fileinto";
fileinto "$U";
msconfig# write
```

`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, add the `destinationspamfilterX` option in `spam` options to 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 destinationspamfilterloptin 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_configX_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 MTA options:

```
msconfig
msconfig> set spamfilter1_config_file
/opt/sun/comms/messaging/config/spamassassin.opt
msconfig# set spamfilter1_library
/opt/sun/comms/messaging/lib/libspamass.so
msconfig# set spamfilter1_optional 1
msconfig# set spamfilter1_string_action=data:,require
["editheader"];addheader "Spam-test" "$U";
msconfig# write
```

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 MTA options:

```
msconfig
msconfig> set spamfilter1_config_file
/opt/sun/comms/messaging/config/spamassassin.opt
msconfig# set spamfilter1_library
/opt/sun/comms/messaging/lib/libspamass.so
msconfig# set spamfilter1_optional 1
msconfig# set spamfilter1_string_action=data:,addtag "[SPAM
detected: $U]";
msconfig# write
```

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 `spamttest` 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 than 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_configX_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. Set the the following MTA options:

```

msconfig
msconfig> set spamfilter1_config_file
/opt/sun/comms/messaging/config/spamassassin.opt
msconfig# set spamfilter1_library
/opt/sun/comms/messaging/lib/libspamass.so
msconfig# set spamfilter1_optional 1
msconfig# spamfilter1_string_action data:, require
["editheader","spamtest"]; \
spamadjust "$U"; addheader "Spam-test" "$U";
msconfig# write

```

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 than 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 `spamassassin.opt` file. You do not have to turn on the channel-specific `master_debug` or `slave_debug`. 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 MTA option `spamfilter1_optional` is set to 1 (highly recommended), 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	<p>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.</p> <p>0 is off, 1 is on, a setting of 2 or greater reports exactly what was received from spamd.</p>	0

field	<p>Specifies the string prefix for the SpamAssassin result. SpamAssassin results look like this:</p> <pre>Spam-test: False ; 0.0 / 5.0 Spam-test: True ; 27.7 / 5.0</pre> <p>The <code>field</code> option provides the means for changing the <code>Spam-test:</code> part of the result. Note that the ":" is removed if an empty <code>field</code> value is specified. If <code>USE_CHECK</code> is set to 0, the result string will look similar to this:</p> <pre>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</pre>	"Spam-test"
host	The name of the system where <code>spamd</code> is running.	localhost
mode	<p>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.</p> <p>0 - Return a <i>verdict string</i> (specified by the <code>verdict</code> option), if the message is spam. The MTA option <code>spamfilterX_string_action</code> can be used to specify what to do if a <i>verdict string</i> is returned. If the <code>verdict</code> option (defined below) is empty or unspecified, and message is spam, a <i>null verdict</i> is returned. The MTA option <code>spamfilterX_null_action</code> can be used to specify what to do if a null verdict is returned. Returns a <i>SpamAssassin default result string</i> if it is not spam. (A default verdict always means to take no action and deliver as normal.)</p> <p>1 - Returns the SpamAssassin <i>result string</i> if the message is found to be spam. Returns a <i>SpamAssassin default result string</i> 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: <pre>True; 6.5 / 7.3</pre></p> <p>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 <code>verdict</code> option is never used.</p> <p>3 - Return the SpamAssassin result string if the message is found to be spam; return the <code>verdict</code> string specified by the <code>verdict</code> option if it is not. You can control the action for the SpamAssassin result string by using the <code>spamfilterX_verdict_n</code> and <code>spamfilterX_action_n</code> matched pair. You can control the action for the <code>verdict</code> string by using <code>spamfilterX_string_action</code>.</p>	0
port	Specifies the port number where <code>spamd</code> listens for incoming requests.	783

USE_CHECK	<p>1 - The <code>spamd CHECK</code> command is used to return the SpamAssassin score.</p> <p>0 - Enables use of the <code>SYMBOLS</code> 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 <i>field</i> above.</p>	
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	<p>Specify the name of a mapping to probe with address information as the plugin receives recipient addresses from the MTA. The probe format is:</p> <pre>current-username current-recipient-address current-optin-string</pre> <p>Both the <i>current-optin-string</i> and the preceding vertical bar are omitted if no optin value was specified.</p> <p>If the mapping sets the <code>\$Y</code> flag the output string is taken to be the updated username to pass to <code>spamd</code>.</p>	""
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
<code>verdict=""</code> (not set)	yes	null	SpamAssassin result	SpamAssassin result	SpamAssassin result
	no	default	default	SpamAssassin result	default
<code>verdict=string</code>	yes	<i>verdict string</i>	SpamAssassin result	SpamAssassin result	SpamAssassin result
	no	default	default	SpamAssassin result	<i>verdict string</i>

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. Set the following MTA options:

```
msconfig
msconfig> set spamfilter1_config_file
/opt/sun/comms/messaging/config/SAVSE.opt
msconfig# set spamfilter1_library
/opt/sun/comms/messaging/lib/libicap.so
msconfig# set spamfilter1_optional 1
msconfig# set spamfilter1_string_action data:,discard
msconfig# write
```

`spamfilter1_config_files` specifies the SAVSE configuration file.

`spamfilter1_library` specifies the location of the SAVSE shared library.

`spamfilter1_optional` set to 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, add the `destinationspamfilterloptin spam` options on the `ims-ms` channel after running `msconfig edit channels`:

```
!  
! 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 destinationspamfilterloptin 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 look something like this:
`/etc/init.d/symcscna start`

Other Possible Configurations

Setting `mode` to 0 can be used with a `spamfilterX_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 `spamfilterX_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 set the following MTA option. Example:

```
msconfig set 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 the `spamfilterX_config_file` MTA option. 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	<p>Specifies the prefix for the ICAP result. SAVSE result strings look like this:</p> <pre>Virus-Test: False Virus-Test: True; W32.Mydoom.A@mm.enc</pre> <p>This option provides a way to change the <code>Virus-Test:</code> part of the result. Note that the ":" is removed if an empty <code>field</code> value is specified.</p>	Virus-Test
host	The name of the system where the ICAP filtering server is running	localhost
mode	<p>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</p> <p>0 - Returns a <i>verdict string</i> (specified by the <code>verdict</code> option), if the message contains a virus. The MTA option <code>spamfilterX_string_action</code> can be used to specify what to do if a <i>verdict string</i> is returned. If the <code>verdict</code> option is empty or not set, a <i>null verdict</i> is returned. The MTA option <code>spamfilterX_null_action</code> 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.</p> <p>1 - Return the ICAP <i>result string</i> 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:</p> <pre>VIRUS TEST: FALSE VIRUS-TEST: TRUE; W32.Mydoom.A@mm.enc</pre> <p>2 - Return an ICAP result string unconditionally; no default or null verdict is ever returned and the <code>verdict</code> 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:</p> <pre>spamfilterX_string_action=data:,require ["editheader"]; addheader "\$U"</pre> <p>3 - Return the ICAP result string if the message is found to contain a virus; return the <code>verdict</code> string specified by the <code>verdict</code> 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 <code>spamfilterX_verdict_n</code> and <code>spamfilterX_action_n</code> matched pair. You can control the action for the <code>verdict</code> string by using <code>spamfilterX_string_action</code>.</p>	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

<i>verdict</i> Setting	Virus?	<i>mode=0</i>	<i>mode=1</i>	<i>mode=2</i>	<i>mode=3</i>
<i>verdict=""</i> (not set)	yes	null	ICAP result	ICAP result	ICAP result
	no	default	default	ICAP result	default
<i>verdict=string</i>	yes	<i>verdict string</i>	ICAP result	ICAP result	ICAP result
	no	default	default	ICAP result	<i>verdict string</i>

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, `clamscan` 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. Set the following MTA options:

```

msconfig
msconfig> set spamfilter2_config_file
/opt/sun/comms/messaging/config/clamav.opt
msconfig# set spamfilter2_library
/opt/sun/comms/messaging/lib/libclamav.so
msconfig# set spamfilter2_string_action data:,require ["jettison"];
jettison;
msconfig# write

```

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, add the destinationspamfilterXoptin virus options on the `ims-ms` channel after running `msconfig edit channels`:

```

!
! 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`.) 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:

```

X50!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 consists 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	<p>Enables or disables debug output from the ClamAV interface module. (Debug output from <code>clamd</code> itself is controlled by options on the <code>clamd</code> command line.) The larger the value, the more debugging output will be produced.</p> <p>0 produces no output. 1 provides basic debugging. 2 adds logging of TCP traffic from <code>clamd</code>.</p>	0
FIELD	<p>Specifies the ClamAV result string prefix. ClamAV result strings generally look something like one of the following:</p> <pre>Virus-Test: False Virus-Test: True ; Worm.Mydoom.I</pre> <p>The <code>FIELD</code> option provides the means for changing the <code>Virus-Test</code> part of the result. Note that the <code>": "</code> will also be removed if an empty <code>FIELD</code> value is specified.</p>	"Virus-Test"
USE_INSTREAM	<p>(New in MS7u2) Enables (1) or disables (0) the use of <code>INSTREAM</code> scanning in <code>clamd</code>. Starting with ClamAV 0.95, <code>clamd</code> 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.</p>	0
MESSAGE_BUFFER_SIZE	<p>Due to the nature of the <code>clamdsan/clamd</code> 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.</p>	1048576

MODE	Controls the translation of ClamAV results to verdict information. Four different modes are available: 0 - Return the verdict string specified by the <code>VERDICT</code> 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 <code>VERDICT</code> 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 <code>VERDICT</code> 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 <code>VERDICT</code> option if it is not.	0
PORT	Specifies the port <code>clamd</code> is running on.	3310
SOCKS_HOST	Specifies the name of an intermediate SOCKS server. If this option is specified the <code>clamd</code> 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`, `spamttest` 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. `spamttest` 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.

`spamttest` 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 `spamttest` forces the score into an integer value between 0 and 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 `spamttest` test sees. `:percent` is supported by `spamttest` (see [SIEVE Email Filtering: Spamttest and Virustest Extensions draft-ietf-sieve-spamttestbis-05](#)). Note that `spamttest` can 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 `spamttest` extension should be listed instead.

For example, a possible use of `spamadjust` with a SpamAssassin `MODE` setting of 2 might be:

```
msconfig set spamfilterX_string_action data:,require ["spamttest"];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 "spamttest";
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 ["spamttest", "relational",
"comparator-i;ascii-numeric", "fileinto"];
if anyof (address :matches "from" ["*@siroe.com", "*@*.siroe.com"])
{keep;}
elsif spamttest :value "ge" :comparator "i;ascii-numeric" "8"
{discard;}
elsif spamttest :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](#)
- [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
<code>\${auth_authen}</code> (MAIL FROM)	Authenticated sender address.
<code>\${auth_author}</code> (MAIL FROM)	The value of the AUTH parameter to MAIL FROM.
<code>\${client_addr}</code> (CONNECT)	The IP address of the SMTP client, expressed as a dotted quad value. Only set when SMTP over TCP is being used.
<code>\${destination_channel}</code> (RCPT TO)	MTA destination channel for the current recipient.
<code>\$i</code> (MAIL FROM)	Queue ID for the current message. The MTA generates a unique ID for each session. This ID is what appears in the <code>\$i</code> macro.
<code>\$j</code> (CONNECT)	Text placed in the "by" clause of Received: header fields. This is controlled by the <code>RECEIVED_DOMAIN</code> MTA option. If the option is not set the official host on the local channel is used instead.
<code>\${mail_addr}</code> (MAIL FROM)	The MAIL FROM address for the current transaction.
<code>\${mail_host}</code> (MAIL FROM)	The host part of the MAIL FROM address for the current transaction.
<code>\${optin}</code> (RCPT TO)	Spamfilter optin value for the current RCPT TO address.
<code>\${rcpt_addr}</code> (RCPT TO)	Current RCPT TO address.
<code>\${rcpt_host}</code> (RCPT TO)	The host part of the current RCPT TO address.
<code>\${rcpt_mailer}</code> (RCPT TO)	Set to "local" for valid recipient addresses, "error" for invalid addresses.
<code>\${source_channel}</code> (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 <code>libmilter</code> 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.

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:

```
msconfig
msconfig> set spamfilter1_library
/opt/sun/comms/messaging/lib/libmilter.so
msconfig# set spamfilter1_config_file
/opt/sun/comms/messaging/lib/milter.opt
msconfig# write
```

- 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 `spamfilterX_string_action` MTA option:
`msconfig set spamfilterX_string_action data:,$M`
This MTA option setting is used unconditionally, but it must be set 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
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
SOCKS_HOST	String . Specifies the name of an intermediate SOCKS server. If this option is specified the Milter connection is made through the specified SOCKS server and not directly.	""
SOCKS_PORT	Integer. Specifies the port the intermediate SOCKS server is running on.	1080

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](#)
- [Fource "detour" routing of hosted users \(`aliasdetourhost`, `aliasoptindetourhost`\)](#)
- ["Handling Forged Email by Using the Sender Policy Framework" in *Messaging Server System Administrator's Guide*](#)
- [Mail Filtering and Access Control in Unified Configuration](#)
- [Configuring Client Access to POP, IMAP, and HTTP Services in Unified Configuration](#)
- [Verify that the domain on the MAIL FROM: line is in the DNS \(`mailfromdnsverify`, `nomailfromdnsverify`\)](#)
- [Expansion of multiple addresses \(`expandlimit`, `expandchannel`, `holdlimit`\)](#)
- [To Create MTA-Wide Filters](#)
- [Configuring SMTP Relay Blocking](#)

Denial of Service Technologies:

- [Using and Configuring MeterMaid for Access Control](#)

- [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` `rule`set 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 11. JMQ Notification in Unified Configuration

JMQ Notification in Unified Configuration

Oracle Communications Messaging Server can use Oracle GlassFish Server 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.

Topics:

- [Setting JMQ Notifications in Unified Configuration](#)
- [Where to Go for More Information](#)

Setting JMQ Notifications in Unified Configuration

To set JMQ notifications in legacy configurations, you use the string `libjmqnotify` in the `local.store.notifyplugin` option. For example, the following command configures the instance name `jmq1`:

```
./configutil -o local.store.notifyplugin -v 'libjmqnotify$jmq1$'
```

In Unified Configuration, the `local.store.notifyplugin` option is replaced with the `notifytarget:*_notifytype` option, where `*` is the instance name. Thus, for an instance name `jmq1` that uses JMQ as its event notification service, you would set `notifytarget:jmq1_notifytype` to `jmq` as in the following example:

```
./msconfig
msconfig> set notifytarget:jmq1_notifytype jmq
msconfig# write -remark "notifytarget jmq1"
msconfig> quit
```



Note

In Messaging Server 7 Update 5, the `libibiff` and `libjmqnotify` notification plugins are now built-in.

Where to Go for More Information

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 in Unified Configuration \(Example\)](#)

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 in Unified Configuration](#)
- [Configuring a JMQ Notification Service in Unified Configuration \(Tasks and Examples\)](#)
- [JMQ Notification Messages and Properties in Unified Configuration \(Reference\)](#)

Configuring a JMQ Notification Service in Unified Configuration (Tasks and Examples)

Configuring a JMQ Notification Service in Unified Configuration (Tasks and Examples)

This information describes how to configure a JMQ notification plug-in.

Topics:

- [Planning for Your JMQ Notification Service](#)
- [To Configure an Instance of the JMQ Notification Plug-in](#)
- [Specifying Notification Messages that Use More Than One `msconfig` Option](#)
- [To Configure New-Message and Updated-Message Notifications with Message Headers and Message Bodies](#)
- [To Configure Deleted-Message Notifications with Message Headers](#)
- [Configuring Notifications for Changes in Message Status](#)

For related topics on JMQ notification, see the following pages:

- [Enabling JMQ Notification in Unified Configuration \(Example\)](#)
- [JMQ Notification Overview in Unified Configuration](#)
- [JMQ Notification Messages and Properties in Unified Configuration \(Reference\)](#)

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 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*.
[Sun GlassFish Message Queue 4.4 Update 1 Installation Guide](#).
For information about configuring and deploying Message Queue, see the *Sun GlassFish Message Queue 4.4 Administration Guide*.
3. Write one or more Message Queue clients that 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 GlassFish Message Queue 4.4 Developer's Guide for Java Clients*.

4. Configure and enable the JMQ notification plug-in for producing notification messages. The remainder of this information 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 GlassFish Message Queue 4.4 Administration Guide](#).

To Configure an Instance of the JMQ Notification Plug-in



Note

In Messaging Server 7 Update 5, the `libibiff` and `libjmqnotify` notification plugins are now built-in.

In this procedure, you first configure the message events that 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:

```
msconfig set notifytarget:<instance>.notifytype jmq
```

Before You Begin

You should install, configure, and deploy the following products:

- Messaging Server
- Oracle GlassFish Message Queue (see the Unified Communications Suite Release Notes in *Unified Communications Suite Installation and Configuration Guide* for more information)



Note

Most of the configuration options that you configure in the following steps are optional. For a list of their default values, see the [Messaging Server Reference](#).

The `msconfig` options used by the default instance have names of the following form:

```
notifytarget:<instance>.<option>
```

1. Configure the notification event message parameters. For each kind of notification event message you want to include in the instance, use the `notifytarget:instance.event}` option. For example, to enable notifications for new messages, type:

```
msconfig set notifytarget:<jmqnotify>.newmsg 1
```

where `jmqnotify` is the name of the default instance and `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](#).

A few notification messages use more than one option 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 options 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:

```
msconfig set notifytarget:jmq1.newmsg 1
msconfig set notifytarget:jmq2.newmsg 1
```

2. Specify the host where the Message Queue destination (broker) is running.
For example:

```
msconfig set notifytarget:<instance>.jmqhost "127.0.0.1"
```

3. Specify the port for the Message Queue broker.
For example:

```
msconfig set notifytarget:<instance>.jmqport "7676"
```

4. Specify the user ID of the Message Queue user authorized to produce messages to the service.
For example:

```
msconfig set notifytarget:<instance>.jmquser "guest"
```

5. Specify the password of the Message Queue user.
You need to do this in interactive mode, for example:

```
./msconfig
msconfig> set -prompt notifytarget:<instance>.jmqpwd
Password:
Verify:
msconfig# write
msconfig> exit
```

6. Specify whether the destination is a "topic" or a "queue".
For example:

```
msconfig set notifytarget:<instance>.destinationtype "queue"
```

7. Specify the destination *topic* or *queue* name.
For example, type one of the following commands:

```
msconfig set notifytarget:<instance>.jmqqueue "JES-MS"
```

or

```
msconfig set notifytarget:<instance>.jmqtopic "JES-MS"
```

The `jmqqueue` and `jmqtopic` options 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 are sent.

8. Specify the Message Queue priority assigned to messages produced by this instance.

For example:

```
msconfig set notifytarget:<instance>.priority 3
```

The default value of the `priority` option is 4.

9. Specify the length of time (in milliseconds) that messages are retained by the Message Queue broker.

For example:

```
msconfig set notifytarget:<instance>.ttl 100
```

This 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.

10. Specify the message persistence.

For example:

```
msconfig set notifytarget:<instance>.persistent 1
```

The 1 specifies that persistent messages are used in the Message Queue service. Allowed values are 1 (persistent) and 0 (non-persistent).

11. Configure the instance name.

To configure a single instance with the default name, type the following:

```
msconfig set notifytarget:<instance>.notifytype jmq
```

Note that `jmqnotify` is the default instance name.

To configure a different instance name such as `jmq42`, type the following command:

```
msconfig set notifytarget:jmq42.notifytype jmq
```

The `msconfig` options read by the `jmq42` instance would have names like:

```

notifytarget:jmq42.<option>
{code:none}

h2. To Configure Multiple Instances

# 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 options:
{code:none}
msconfig set notifytarget:jmq1.newmsg 1
msconfig set notifytarget:jmq2.newmsg 1
msconfig set notifytarget:jmq2.purgemsg 1

```

You also must specify options 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](#).

1. Configure the instance names.

To configure two instances named `jmq1` and `jmq2`, type the following command:

```

msconfig set notifytarget:jmq1.notifytype jmq
msconfig set notifytarget:jmq2.notifytype jmq

```

In this example, the first instance builds its configuration from options with the name `jmq1`:

```

notifytarget:jmq1.<option>

```

The second instance builds its configuration from options with the name `jmq2`:

```

notifytarget:jmq2.<option>

```

Specifying Notification Messages that Use More Than One `msconfig` Option

For most notification messages, you specify the message by running a single `msconfig notifytarget` command.

However, the following notification messages are (or can be) configured with more than one `msconfig notifytarget` command:

- `newmsg`
- `updatemsg`
- `deletemsg`
- `msgflags`

The following procedures 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:

```
msconfig set notifytarget:<instance>.newmsg 1
msconfig set notifytarget:<instance>.updatemsg 1
```

2. Specify the `maxheadersize` parameter with a value greater than zero:

```
msconfig set notifytarget:<instance>.maxheadersize 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:

```
msconfig set notifytarget:<instance>.maxbodysize 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:

```
msconfig set notifytarget:<instance>.deletemsg.enable 1
```

2. Specify the `expungemsg` option:

```
msconfig set notifytarget:<instance>.expungemsg 1
```

Setting the value to 1 enables message headers to be carried with deleted-message notifications. The default value of `expungemsg` 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:

```
msconfig set notifytarget:<instance>.maxheadersize 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.

A message-flag notification is produced whenever a status flag has changed because the email message was:

- Answered
- Flagged
- Deleted
- Seen (read)
- Draft

When a message-flag 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](#).

To Enable Notifications When Message-Flags Have Changed

To enable message-flag notifications, you must configure the following `msconfig` option:

- `notifytarget:<instance>.msgflags 1`

This option 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 option is instance specific. If an instance does not use message-flag notifications, be sure that this option is disabled (default).

To Enable Conditional Notifications for Specified Users

This task enables 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:

```
msconfig set notifytarget:<instance>.enable 0
```

2. Specify the name of the LDAP attribute which, if present in the user's LDAP entry, enables notifications for that user and specify the instance to use:

```
msconfig set notifytarget:<instance>.ldapdestination  
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.

4. Ensure that this LDAP attribute is cached in enqueued messages and carried in LMTP deliveries.

```
msconfig set LDAP_SPARE_1=mailEventNotificationDestination  
msconfig set 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

This task enables 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:

```
msconfig set notifytarget:<instance>.enable 0
```

2. Specify the name of the LDAP attribute which, if present in the user's LDAP entry, enables notifications for that user and specify the instance to use:

```
msconfig set notifytarget:<instance>.ldapdestination  
mailEventNotificationDestination
```

3. Add the attribute to the LDAP entries of the users who require notifications. 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 options that you set to provide configuration information needed by Message Queue. These options are listed in the next step.

4. Use the `msconfig set` command to configure the following JMQ configuration options:

```
notifytarget:<jmq_destination_name>.jmqhost  
notifytarget:<jmq_destination_name>.jmqport  
notifytarget:<jmq_destination_name>.jmquser  
notifytarget:<jmq_destination_name>.jmqpwd  
notifytarget:<jmq_destination_name>.destinationtype  
  
notifytarget:<jmq_destination_name>.jmqtopic  
or  
notifytarget:<jmq_destination_name>.jmqqueue  
  
notifytarget:<jmq_destination_name>.priority  
notifytarget:<jmq_destination_name>.ttl  
notifytarget:<jmq_destination_name>.persistent
```

For example:

```
msconfig set notifytarget:mqdestination1.jmqhost "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 that 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 go to the second destination, add:

```
mailEventNotificationDestination: messagequeuedestination2
```

For all users whose notifications go to the third destination, add:

```
mailEventNotificationDestination: messagequeuedestination3
```

- b. Set each JMQ configuration parameter (such as `jqmhost`) to its specific value. For example:

```
msconfig set notifytarget:mqdestination2.jmqhost "127.0.0.2"
```

```
msconfig set notifytarget:jmqdestination3.jmqhost "127.0.0.3"
```

and so on for each parameter.

6. Restart Messaging Server.

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

Enabling JMQ Notification in Unified Configuration (Example)

Enabling JMQ Event Notification in Unified Configuration (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 in Unified Configuration](#)
- [Configuring a JMQ Notification Service in Unified Configuration \(Tasks and Examples\)](#)
- [JMQ Notification Messages and Properties in Unified Configuration](#)

Event Notifications in Messaging Server Overview

Messaging Server supports 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.

Enable Java Message Queue (JMQ)

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. Reset the `admin/guest` password and add the `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

1. Enable the appropriate Messaging Server settings.
In this example, the target is jmqnotify.

```

cd /opt/sun/comms/messaging64/sbin
./msconfig set notifytarget:jmqnotify.notifytype jmq
./msconfig set notifytarget:jmqnotify.newmsg 1
./msconfig set notifytarget:jmqnotify.updatemsg 1
./msconfig set notifytarget:jmqnotify.deletemsg 1
./msconfig set notifytarget:jmqnotify.maxheadersize 1024
./msconfig set notifytarget:jmqnotify.jmqhost "127.0.0.1"
./msconfig set notifytarget:jmqnotify.jmqport "7676"
./msconfig set notifytarget:jmqnotify.jmquser "jesuser"
./msconfig set notifytarget:jmqnotify.destinationtype "queue"
./msconfig set notifytarget:jmqnotify.jmqqueue "jesms"
./msconfig set notifytarget:jmqnotify.priority 3
./msconfig set notifytarget:jmqnotify.ttl 1000
./msconfig set notifytarget:jmqnotify.persistent 1

```

2. Set the password for jmquser option, which in this example, is jesuser.

```

./msconfig
msconfig> set -prompt notifytarget:jmqnotify.jmqpwd
Password:
Verify:
msconfig# write
msconfig> exit

```

3. Restart Messaging Server.

```

./stop-msg
./start-msg

```

4. Verify that the configuration is working by checking for an entry such as the following in the *msg-srv-base/log/imap* log:

```

[29/Oct/2012:15:45:38 -0700] ipg-test3 imapd[18062]: General
Notice: JMQ notifications enabled: jmqnotify

```

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
Querying the broker specified by:

-----
Host          Primary Port
-----
localhost     7676

Version              4.4 Update 1
Instance Name        imqbroker
Broker ID
Primary Port         7676
Broker is Embedded   false
Instance Configuration/Data Root Directory /var/imq

Current Number of Messages in System      36
Current Total Message Bytes in System     19692

Current Number of Messages in Dead Message Queue  0
Current Total Message Bytes in Dead Message Queue  0

Log Dead Messages                false
Truncate Message Body in Dead Message Queue      true

Max Number of Messages in System          unlimited (-1)
Max Total Message Bytes in System         unlimited (-1)
Max Message Size                        70m

Auto Create Queues                  true
Auto Create Topics                   true
Auto Created Queue Max Number of Active Consumers unlimited (-1)
Auto Created Queue Max Number of Backup Consumers  0

Cluster ID
Cluster is Highly Available           false
Cluster Broker List (active)
mq://10.133.153.173:7676/
Cluster Broker List (configured)
Cluster Master Broker
Cluster URL

Log Level                            WARNING
Log Rollover Interval (seconds)       604800
Log Rollover Size (bytes)             268435456

Successfully queried the broker.
```

You can also run this command:


```

# imqcmd -u admin list dst
Listing all the destinations on the broker specified by:

-----
Host          Primary Port
-----
localhost     7676

-----
Name          Type          State          Producers          Consumers          Msgs
              UnAck         Avg Size       Total Wildcard    Total Wildcard    Count  Remote
-----
Queue        RUNNING    24           -                   0                   -                   0                   0                   0                   0.0
mq.sys.dmq   Queue     RUNNING    0                   -                   0                   -                   0                   0
0            0.0

Successfully listed destinations.

```

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](#)

JMQ Notification Messages and Properties in Unified Configuration

JMQ Notification Messages and Properties in Unified Configuration

This page contains the following topics:

- [Notification Messages](#)
- [Rules and Guidelines for Notification Messages](#)
- [Notifications for Particular Message Types](#)
- [Default Values of the `notifytarget` Options](#)
- [Notification Message Properties](#)

For related topics on JMQ notification, see the following pages:

- [Enabling JMQ Notification in Unified Configuration \(Example\)](#)
- [JMQ Notification Overview in Unified Configuration](#)
- [Configuring a JMQ Notification Service in Unified Configuration \(Tasks and Examples\)](#)

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.

An `msconfig` option specifies each kind of message to be produced. You determine which events generate messages by configuring various options. The options 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 options that enable these messages, see the [notifytarget Options and Their Default Values](#) table.

JMQ Notification Messages

Notification Message	Description
deletemsg	Messages marked as "Deleted" are removed from the mailbox. This is the equivalent to IMAP expunge.
expungemsg	Messages are expunged from the mailbox.
login	User logged in from IMAP, HTTP, or POP. (This message is enabled with the <code>notifytarget:*.loguser</code> option.)
logout	User logged out from IMAP, HTTP, or POP. (This message is enabled with the <code>notifytarget:*.loguser</code> option.)
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 is expunged.
purgemsg	Message expunged (as a result of an expired date) from the mailbox by the server process <code>imexpire</code> . This is a server side expunge, whereas <code>deletemsg</code> 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 `expungemsg` 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` messages 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 option allows for `INBOX` only (value = 0), or for both the `INBOX` and all other folders (value = 1):

```
notifytarget:<jmqnotify>.noninbox
```

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 options such as `store.messageType.enable`. For information about configuring and managing message types, see [Managing Message Types in the Message Store in Unified Configuration](#).

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 `notifytarget` 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 `notifytarget` configuration is the following:

```
notifytarget = /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 options for that base instance would be of the form:

```
notifytarget.msgjmqnotify.<option-name> = <option-value>
```

To override the notifytarget handling of type 1 messages, use options of the form:

```
notifytarget.msgjmqnotify.1.<option-name> = <option-value>
```

Even though there is no "msgjmqnotify.1" instance created explicitly in the notifytarget option, it is created automatically from the "msgjmqnotify" instance because `messagetype 1` is defined. The `notifytarget.msgjmqnotify.1.option-name` settings are used to override the corresponding options from the `notifytarget.msgjmqnotify.option-name` settings.

Default Values of the notifytarget Options

The notification messages and the configuration information needed by Message Queue are configured with the options shown in the following table. For complete definitions of the options, see http://msg.wikidoc.info/index.php/Configutil_Reference.

notifytarget Options and Their Default Values

Option	Default Value
notifytarget:*.annotatmsg	0 — Disabled
notifytarget:*.changeflag	0 — Disabled
notifytarget:*.copymsg	0 — Disabled
notifytarget:*.deletmsg	1 — Enabled
notifytarget:*.destinationtype	"topic"
notifytarget:*.enabled	1 — Turned on by default
notifytarget:*.expungmsg	0 — Disabled
notifytarget:*.jmqhost	"127.0.0.1"
notifytarget:*.jmqport	7676
notifytarget:*.jmqpwd	"guest"
notifytarget:*.jmqqueue	"JES-MS"
notifytarget:*.jmqtopic	"JES-MS"
notifytarget:*.jmquser	"guest"
notifytarget:*.ldapdestination	null — Turned off by default
notifytarget:*.loguser	1 — Enabled
notifytarget:*.maxbodysize	0 — Disabled
notifytarget:*.maxheadersize	0 — Disabled
notifytarget:*.msgflags	0 — Disabled
notifytarget:*.msgtypes	0 — Disabled
notifytarget:*.newmsg	1 — Enabled
notifytarget:*.noninBox	0 — Disabled
notifytarget:*.overquota	1 — Enabled
notifytarget:*.persistent	1 — Enabled
notifytarget:*.priority	4
notifytarget:*.purgmsg	1 — Enabled
notifytarget:*.readmsg	1 — Enabled
notifytarget:*.ttl	0 — Indicates that messages never time out
notifytarget:*.underquota	1 — Enabled
notifytarget:*.updatmsg	1 — Enabled

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

The following table describes the standard notification message properties. These properties are present in all JMS messages.

Standard Notification Message Properties

Property	Data Type	Description
<code>hostname</code>	<code>ConstMQString</code>	The host name of the machine that generated the message.
<code>pid</code>	<code>MQInt32</code>	ID of the process that generated the message.
<code>process</code>	<code>ConstMQString</code>	Specifies the name of the process that generated the message.
<code>timestamp</code>	<code>MQFloat64</code>	Specifies the number of milliseconds since the epoch (midnight GMT, January 1, 1970).

Properties Specific to Particular Notification Messages

The following table 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 the [Properties Carried with Each Notification Message](#) table.

Properties Specific to Particular Notification Messages

Property	Data Type	Description
<i>attr</i> <i>n</i>	ConstMQString	Annotation attribute name.
<i>client</i>	ConstMQString	The IP address of the Message Queue client associated with the message.
<i>diskquota</i>	MQInt32	The disk space quota, in kilobytes, for the user associated with the message. The value is set to -1 to indicate no quotas.
<i>diskquotaused</i>	MQInt32	The amount of disk space used by the user associated with the message, in kilobytes.
<i>entry</i> <i>n</i>	ConstMQString	Annotation entry name.
<i>hdrLen</i>	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.
<i>imapUid</i>	MQInt32	The IMAP uid property associated with the message.
<i>lastUid</i>	MQInt32	The last IMAP uid value used in the mailbox.
<i>mailboxName</i>	ConstMQstring	The message-store mailbox name associated with the event. The <i>mailboxName</i> has one of the following formats (where <i>uid</i> is the user's unique identifier): <i>uid</i> — identifies the inbox of a user in the default (primary) domain. <i>uid@domain</i> — identifies the inbox of a user in a hosted domain. <i>uid/mailboxname</i> — identifies the top-level mailbox of a user in the default domain. <i>uid@domain/mailboxname</i> — identifies the top-level mailbox of a user in a hosted domain. <i>uid/foldername/mailboxname</i> — identifies a mailbox in a folder of a user in the default domain. <i>uid@domain/foldername/mailboxname</i> — identifies a mailbox in a folder of a user in a hosted domain.
<i>msgflags</i>	ConstMQString	List of current message flags.
<i>msgquota</i>	MQInt32	The user's quota for the maximum number of messages. The value is set to -1 to indicate no quotas.
<i>newflags</i>	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 <i>oldflags</i> , when a <i>MsgFlags</i> notification message is produced. For the syntax and values for <i>newflags</i> , see Syntax for newflags and oldflags Properties , below this table.
<i>numDeleted</i>	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 <i>DeleteMsg</i>).
<i>numDeleted</i> <i>nn</i>	MQInt32	The total number of messages in the mailbox marked as deleted, specified for each message type. If message types are configured, a <i>numDeleted</i> <i>nn</i> property carries a count for each message type <i>nn</i> . The <i>numDeleted</i> 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: <i>numDeleted</i> =20 <i>numDeleted</i> 3=10 <i>numDeleted</i> 16=7
<i>numMsgs</i>	MQInt32	The total number of messages now in the mailbox.

numMsgs <i>nn</i>	MQInt32	The total number of messages now in the mailbox, specified for each message type. If message types are configured, a numMsgs <i>nn</i> property carries a count for each message type <i>nn</i> . 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=20 numMsgs3=10 numMsgs16=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 <i>nn</i>	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 <i>nn</i> property carries a count for each message type <i>nn</i> . 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=20 numSeen3=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 <i>nn</i>	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 <i>nn</i> property carries a count for each message type <i>nn</i> . 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=10 numSeenDeleted16=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

The following table 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).

Properties Carried with Each Notification Message

Property	newmsg, copymsg, updatemsg	read msg, trash purge msg	delete msg flags	login, logout	over quota, under quota	change flag	annotate msg	expunge msg	create	delete	rename
<code>attrn</code>	No	No	No	No	No	No	Yes	No	No	No	No
<code>client</code>	No	No	No	Yes	No	No	No	No	No	No	No
<code>disk quota</code>	No	No	No	No	Yes	No	No	No	No	No	No
<code>disk quota used</code>	No	No	No	No	Yes	No	No	No	No	No	No
<code>entryn</code>	No	No	No	No	No	No	Yes	No	No	No	No
<code>fromUid Validity</code>	Yes	No	No	No	No	No	No	No	Yes	No	Yes
<code>hdrLen</code>	Yes	No	No	Yes	No	No	No	No	No	No	No
<code>hostname</code>	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
<code>imapUid</code>	Yes	No	Yes	Yes	No	No	Yes	No	No	No	No
<code>lastUid</code>	No	No	Yes	No	No	No	No	Yes	No	No	No
<code>mailbox Name</code>	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes
<code>modseq</code>	No	No	No	No	No	Yes	Yes	Yes	No	No	No
<code>msgflags</code>	No	No	No	No	Yes	Yes	No	No	No	No	No
<code>msgquota</code>	No	No	No	No	Yes	No	No	No	No	No	No
<code>newflags</code>	No	No	Yes	No	No	No	No	No	No	No	No
<code>newName</code>	No	No	No	No	No	No	No	No	No	No	Yes
<code>numDeleted</code>	Yes	Yes	Yes	No	No	No	No	No	No	No	No
<code>numDeleted n</code>	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No
<code>numMsgs</code>	Yes	Yes	Yes	No	No	Yes	No	Yes	No	No	No
<code>numMsgsn</code>	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No
<code>numSeen</code>	Yes	Yes	Yes	No	No	No	No	No	No	No	No
<code>numSeenn</code>	Yes*	Yes*	Yes*	No	No	No	No	No	No	No	No

numSeen Deleted	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No
numSeen Deleted <i>n</i>	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
uid Validity	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



Note

The `numDeletedn`, `numMsgsn`, `numSeenn`, and `numSeenDeletedn` properties are carried with notifications only if message types are defined in the message store.

JMQ Notification Overview in Unified Configuration

JMQ Notification Overview in Unified Configuration

Topics:

- [Two Notification Messaging Services](#)
- [Notification Plug-ins](#)
- [Benefits of Using JMQ Notification](#)

For more information JMQ notification, see the following topics:

- [Enabling JMQ Notification in Unified Configuration \(Example\)](#)
- [Configuring a JMQ Notification Service in Unified Configuration \(Tasks and Examples\)](#)
- [JMQ Notification Messages and Properties in Unified Configuration](#)

The Messaging Server notification plug-in enables 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 GlassFish Message Queue 4.4 Update 1](#)
- [Event Notification Service](#)

The Message Queue service 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. Message Queue 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 information 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 enables 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 [Administering Event Notification Service in Messaging Server for Unified Configuration](#).

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 sections describe these benefits in more detail:

- [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.

Starting with Messaging Server 7.5, the `libibiff` and `libjmqnotify` notification plugins are now built-in. In Unified Configuration, you use the `mconfig` command to specify options 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 configuration options.

The options 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 [Configuring a JMQ Notification Service in Unified Configuration \(Tasks and Examples\)](#).

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 12. LMTP Delivery in Unified Configuration

LMTP Delivery in Unified Configuration

This information provides an overview of the Local Mail Transfer Protocol (LMTP), and describes how to configure both the LMTP client and server.

Topics:

- [Overview of LMTP](#)
- [LMTP Delivery Features](#)
- [Messaging Processing in a Two-Tiered Deployment Without LMTP](#)
- [Messaging Processing in a Two-Tiered Deployment With LMTP](#)
- [LMTP Architecture](#)
- [Configuring LMTP](#)
- [LMTP Protocol as Implemented](#)

Overview of LMTP

The Oracle Communications Messaging Server MTA can use Local Mail Transfer Protocol (LMTP), as defined in RFC 2033, to deliver messages to the message store in a multi-tiered Messaging Server deployment. In this scenario, the front-end 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-tiered deployments are used.



Note

LMTP is recommended for use in multi-tiered deployments. Also, the Messaging Server's LMTP service as implemented is not designed to work with third-party LMTP servers or third-party LMTP clients.

LMTP Delivery Features

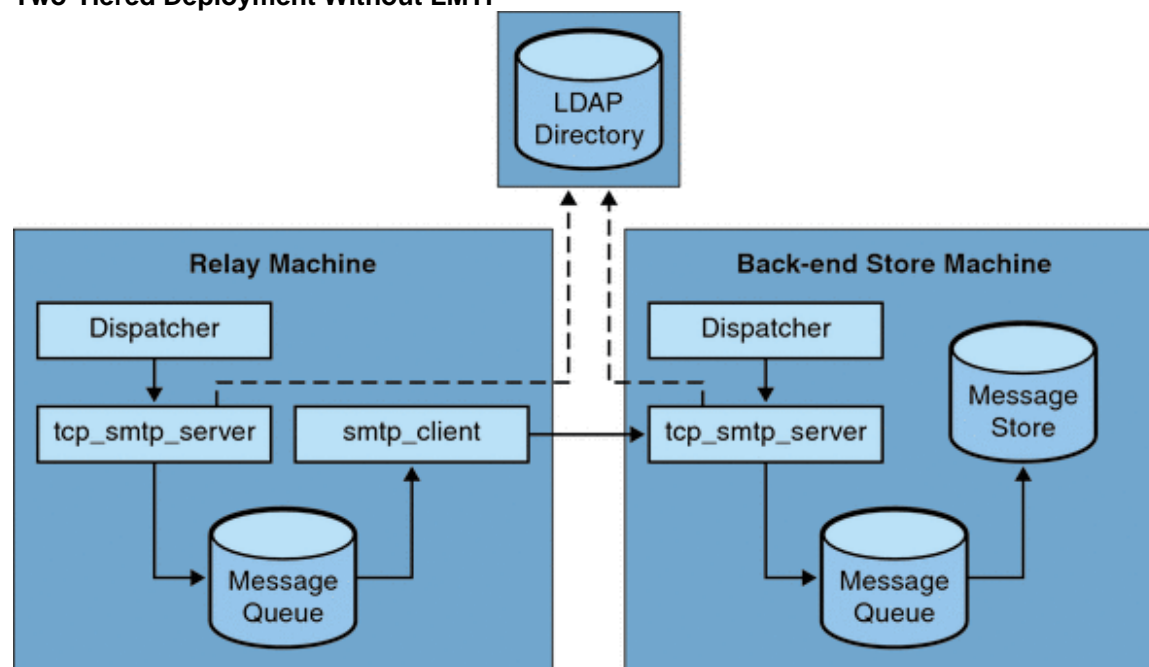
The MTA's LMTP server is more efficient for delivering to the back-end message store because it:

- 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 administering a messaging deployment.

Messaging Processing in a Two-Tiered Deployment Without LMTP

The following figure shows message processing in a two-tiered deployment without LMTP.

Two-Tiered Deployment Without LMTP



Without LMTP, in a two-tiered deployment with relays "in front" of the store systems, inbound message processing 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:

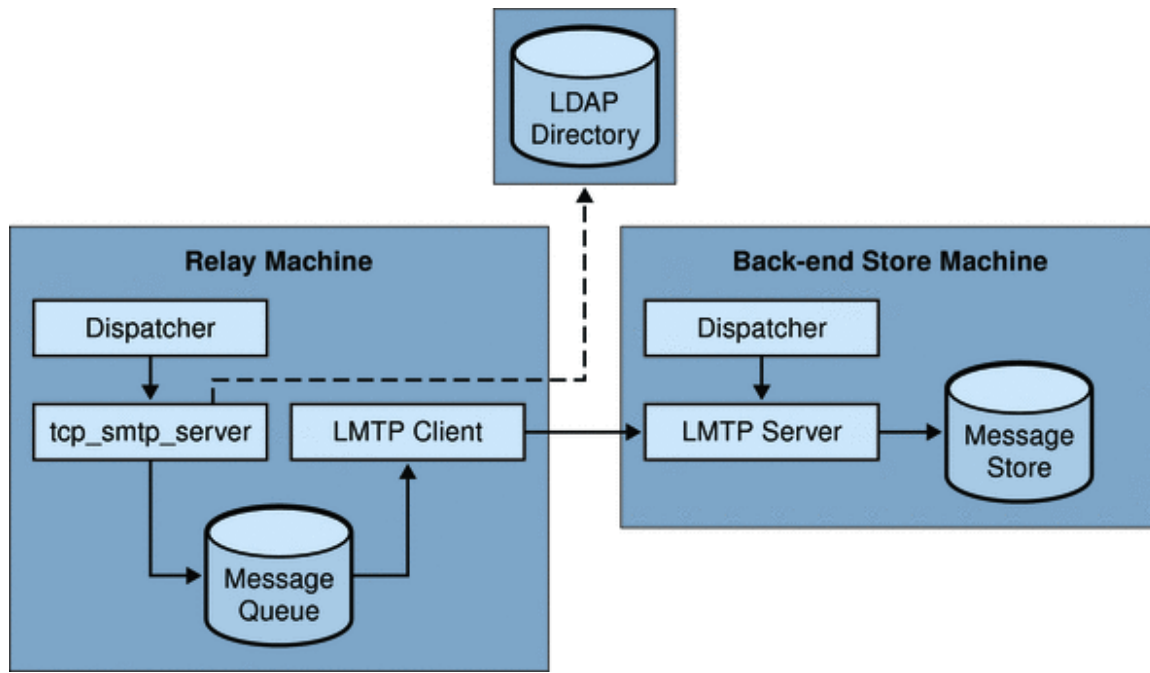
- Looking up the user in the directory
- Determining if the user is within a domain hosted by this email deployment
- Determining if the user is a valid user in the domain
- Rewriting the envelope address to direct the message to the `ims_ms` channel
- Enqueuing the message for delivery to the store

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-Tiered Deployment With LMTP

The following figure shows message processing in a two-tiered deployment scenario with LMTP.

Two-Tiered Deployment With LMTP



With LMTM 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 LMTM port is received by the dispatcher and handed off to the `lmtm_server` process. The LMTM 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 LMTM process on the message store does not engage any MTA machinery for processing addresses or messages.

LMTM Architecture

An LMTM configuration consists of setting up LMTM channels, one for the LMTM client and one for the LMTM server. LMTM channels are special cases of TCP/IP channels. The general SMTP-over-TCP/IP channel is often configured for bidirectional use. An LMTM channel, on the other hand, is configured to be dedicated for either client or server use. You configure the LMTM client channel on the MTA front-end relay host and the LMTM server channel on the back-end message store. Running `commpkg install` on both the front-end relay and back-end store system will configure LMTM properly.

For the most part, the MTA itself can be basically absent from the back-end server. The following items are the only necessary MTA components on the back end:

- The dispatcher
- The LMTM server
- A simple MTA configuration including `mappings`

The dispatcher must run on the back-end server so that it can start the LMTM servers that run under it. Because the dispatcher and the LMTM 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 enqueueing or dequeuing functions, header processing, or address translations. The front-end 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 that 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.

Configuring LMTP

Setting up LMTP requires that you configure both the front-end relay hosts and back-end message store hosts. On the relays, you must change the `mta.delivery_options` option 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. On the LMTP client, you must configure the job controller to run the LMTP client.

This section describes how to configure the LMTP front ends and back ends by using Unified Configuration recipes. You could also perform the same configuration manually by using the `msconfig` command, however, using recipes is faster and makes the task repeatable. You run one recipe on the LMTP front ends and one on the LMTP back ends. For more information on Unified Configuration recipes, see [Recipe Language](#) and [Using Recipes](#).

Topics in this section:

- [Before You Begin](#)
- [To Configure the Front-end MTA Relay with LMTP](#)
- [To Configure Back-End Stores with LMTP and a Minimal MTA](#)

Before You Begin

- To see if any LMTP options are already enabled on your Messaging Server hosts, run the following command:

```
msconfig show mta.delivery_options -default
```

To Configure the Front-end MTA Relay with LMTP

Use the following [example recipe file](#) to configure the inbound MTA relay for LMTP.

1. Make a copy of the example recipe file and save it as `recipe.rcp` in the `config/recipes` directory.
2. In this example, replace the following items with your site-specific values:
 - `myIP`
 - `myNetmaskbits`
3. To run the recipe, type the following command:

```
cd <msg-srv-base>/sbin
./msconfig run <msg-svr-base>/config/recipes/<recipe_name>
```

4. Recompile if running a compiled configuration.

```
./imsimta cnbuild
```

5. Restart Messaging Server.

```
cd <msg-svr-base>/bin
./stop-msg
./start-msg
```

Example Recipe to Configure LMTP Front-end Relay with LMTP

```
# -*- mode: sieve; -*-
description("frontendMTA to backend LMTP store");
keywords(["frontend", "MTA", "LMTP"]);
#
# Sample recipe for a frontend MMP/MTA to a backend LMTP store
# the corresponding recipe for the backend store/LMTP is
backendLMTP.rcp
#
#####
!!!!!!!!!!!!!!!!!!!!!!
# CHANGE THESE
# !!!!!!!!!!!!!!!!!!!!!!!
#
# constants - supplied input
#
# my IP address
myIP = "10.133.158.10";
# network portion of IP address in number of bits
myNetmaskbits = "8";

#####
configure LMTP frontend
#
#
# disable store
#
set_option("store.enable", "0");
set_option("imap.enable", "0");
set_option("pop.enable", "0");

#
# add to rewrite rules
# .lmtpl $E$F$U%H.lmtpl@lmtplcs-daemon
# .lmtpl $B$F$U%H@$H@lmtplcs-daemon
#
# should really check to see if the rewrite rule exists instead
# of unilaterally appending it
#
```

```

append_rewrites([".lmtplib", "$E$F$U%H.lmtplib@lmtplib-daemon"]);
append_rewrites([".lmtplib", "$B$F$U%H@$H@lmtplib-daemon"]);

#
# add channel tcp_lmtplib
# ,----
# | tcp_lmtplib defragment lmtplib multigate connectcanonical fileinto
@$40:$U+$S@$D f\
# | lagtransfer multigate connectcanonical port 225 nomx sin\
# | gle_sys pool SMTP_POOL dequeue_removeoute
# | lmtplib-daemon
# `----
#
set_option("channel:tcp_lmtplib.connectcanonical");
set_option("channel:tcp_lmtplib.defragment");
set_option("channel:tcp_lmtplib.fileinto", "@$40:$U+$S@$D");
set_option("channel:tcp_lmtplib.lagtransfer");
set_option("channel:tcp_lmtplib.lmtplib");
set_option("channel:tcp_lmtplib.multigate");
set_option("channel:tcp_lmtplib.nomx");
set_option("channel:tcp_lmtplib.pool", "SMTP_POOL");
set_option("channel:tcp_lmtplib.port", "225");
set_option("channel:tcp_lmtplib.single_sys");
set_option("channel:tcp_lmtplib.dequeue_removeoute");
set_option("channel:tcp_lmtplib.official_host_name", "lmtplib-daemon");

#
# ! The modified DELIVERY_OPTIONS which activate LMTP
# ! delivery from a frontend relay to the backend
#
# DELIVERY_OPTIONS=\
#     #*mailbox=@$X.LMTP:$M%$\\$2I$_+$2S@lmtplib-daemon,\
#     #&members=*,\
#     #*native=@$X.LMTPN:$M+$2S@native-daemon,\
#     #*unix=@$X.LMTPN:$M,\
#     #*file=@$X.LMTPN:+$F,\
#     #&@members_offline=*,\
#     #/hold=@hold-daemon:$A,\
#     #program=$M%$P@pipe-daemon,\
#     #forward=**,\
#     #*^!autoreply=$M+$D@bitbucket
#
# NOTE NOTE NOTE - have to escape the backslash in the "mailbox=..."
set_option("mta.delivery_options",
"#*mailbox=@$X.LMTP:$M%$\\$2I$_+$2S@lmtplib-daemon,#&members=*,#*native=@$X.

```

```
\ "write -remark frontendLMTP.rcp\" to write out the changes\n");
```

This recipe performs the following configuration on the LMTP front end:

1. Disables the message store, as it is not needed.
2. Enables the necessary rewrite rules.
The recipe language `append_rewrites` function adds entries to existing rewrite rules, or, if none exists, adds them as new rules.
3. Adds the `tcp_lmtpcs` channel with the appropriate options:
 - `connectcanonical`: Tells the MTA to compare the recipient envelope address domain with the channel host proper names, and if the domain name matches one of the channel's host proper names, then connect to the host name corresponding to that host proper name.
 - `defragment`: Any message/partial messages queued to the channel are placed in the `defragmentation` channel queue instead. Once all the parts have arrived, the message is rebuilt and sent on its way.
 - `fileinto`, value of `@$4O:$U+$S@$D`: Specifies how to alter an address when a Sieve filter "fileinto" action is applied. In `$4O`, the `O` is the capital or majuscule letter "o", not the numeral zero 0. The effect is that the explicit source route to the mailhost should be preserved if present, and the foldername should be inserted as a subaddress into the original address, replacing any originally present subaddress.
 - `flagtransfer`: Enables SMTP client support of the XDFLG private SMTP extension command.
 - `lmtp`: Specifies that channel supports LMTP protocol.
 - `multigate`: Instructs the MTA to route the message to the daemon mailbox specified by the daemon channel option on the system specified in the message's To: address.
 - `nomx`: Disables MX lookups.
 - `pool`, value of `SMTP_POOL`: Specifies the `SMTP_POOL` pool where the jobs are created for this channel.
 - `port`, value of `225`: Specifies dispatcher port number.
 - `single_sys`: Creates a single copy of the message for each destination system (more precisely, each destination domain name) associated with a recipient address.
 - `dequeueremoveoveroute`: Causes source routes to be stripped from envelope recipient addresses when the channel dequeues messages (but after the channel has determined how to route the message).
 - `official_host_name`, value of `lmtpcs-daemon`: Specifies the "name" of the system with which this channel communicates. (In legacy configuration, the official host name is specified as the first name on the second line of a channel definition.)
4. Finally, the delivery options are set, which activates LMTP delivery from the front-end relay to the back-end message store.
In the `*mailbox=@$X.LMTP:$M%$\\$2I$_+$2S@lmtpcs-daemon`, portion of the delivery options, the script "escapes" the forward slash, so that the actual entry in the recipe is as follows:

```
... "#*mailbox=@$X.LMTP:$M%$\\$2I$_+$2S@lmtpcs-daemon, #&members=*, ...
```

Note

You need to add the full `delivery_options` as shown in the example. You cannot 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

Use the following [example recipe file](#) to configure the back-end store for LMTP. This recipe:

- Prompts for `myIP` and `feIPs`, if you do not specify them in the recipe (but does not validate the IP addresses)
 - Checks if the `tcp_lmtpss` channel exists before creating it
 - Checks if the dispatcher group exists before creating it
 - Checks if `PORT_ACCESS` entries exist before creating them
 - Treats `feIPs` as a list to allow multiple front ends
 - Can be run multiple times correctly
1. Make a copy of the example recipe file and save it as `recipe.rcp` in the `config/recipes` directory.
 2. In this example, replace the following items with your site-specific values:
 - `myIP`
 - `feIPs`
 3. To run the recipe, type the following command:

```
cd <msg-srv-base>/sbin
./msconfig run <msg-srv-base>/config/recipes/<recipe_name>
```

4. Recompile if running a compiled configuration.

```
./imsimta cnbuild
```

5. Restart Messaging Server.

```
cd <msg-svr-base>/bin
./stop-msg
./start-msg
```

Example Recipe to Configure LMTP Back-end Store with LMTP

```
# -*- mode: sieve; -*-
description("backend store via LMTP");
keywords(["backend", "store", "LMTP"]);
#
# Sample recipe for a backend store via LMTP
# the corresponding recipe for the frontend MMP/MTA is
# frontendLMTP.rcp
#####
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# CHANGE THESE
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
# constants - supplied input
# if you leave it blank, the script will prompt for it
# sample entry
#myIP = "10.133.152.193";
myIP = "";

#
# list of frontend machines that access this store via LMTP
# if you leave it blank, the script will prompt for it
```

```

# sample entry:
#feIPs = ["10.133.152.192", "10.133.152.193"];
feIPs = [];

#####
prompt for myIP and feIP if needed

if (length(myIP) <= 0) {
    myIP = read("Enter the IP address of this host: ");
}

if (length(feIPs) <= 0) {
    loop {
        ip = read("Enter the IP address of a frontend machine (<RET> if no
more): ");
        exitif (ip == "");
        push(feIPs, ip);
    }
}

#####
configure LMTP backend

#
# create tcp_lmtpss channel. In legacy config, this would show up as:
# tcp_lmtpss lmtp flagtransfer identnonenumeric
# tcp_lmtpss-daemon
if exists_channel("tcp_lmtpss") {
    warn("-- WARNING: tcp_lmtpss channel already exists.");
} else {
    # This creates the channel by using individual options
    #set_option("channel:tcp_lmtpss.flagtransfer");
    #set_option("channel:tcp_lmtpss.identnonenumeric");
    #set_option("channel:tcp_lmtpss.lmtp");
    #set_option("channel:tcp_lmtpss.official_host_name",
"tcp_lmtpss-daemon");
    # an alternative way of doing it as a one-liner
    print("-- INFO: Adding tcp_lmtpss channel\n");
    add_channel("tcp_lmtpss",
        ["flagtransfer", "",
        "identnonenumeric", "",
        "lmtp", "",
        "official_host_name", "tcp_lmtpss-daemon"]);
}

#
# dispatcher.cnf
# uncomment [SERVICE=LMPSS] block
#
if exists_group("dispatcher.service:LMPSS") {
    warn("-- WARNING: dispatcher.service:LMPSS group already exists");
} else {
    print("-- INFO: Creating dispatcher.service:LMPSS group\n");
    # This creates the group by using individual options
    #set_option("dispatcher.service:LMPSS.image",

```

```

"IMTA_BIN:tcp_lmtp_server");
  #set_option("dispatcher.service:LMPSS.logfilename",
"IMTA_LOG:tcp_lmtpss_server.log");
  #set_option("dispatcher.service:LMPSS.parameter",
"CHANNEL=tcp_lmtpss");
  #set_option("dispatcher.service:LMPSS.tcp_ports", "225");
  #set_option("dispatcher.service:LMPSS.stacksize", "2048000");
  #set_option("dispatcher.service:LMPSS.enable", "1");

# alternate way of doing this in one line
add_group("dispatcher.service:LMPSS",
  ["image", "IMTA_BIN:tcp_lmtp_server",
  "logfilename", "IMTA_LOG:tcp_lmtpss_server.log",
  "parameter", "CHANNEL=tcp_lmtpss",
  "tcp_ports", "225",
  "stacksize", "2048000",
  "enable", "1"]);
}

#
# add PORT_ACCESS mapping entries
#
# allow frontends (feIPs) to access LMTP port
# TCP|*|225|10.133.152.192|* $Y
# TCP|*|226|10.133.152.192|* $Y
#
# ! Allow 'msprobe' on this host (myIP) to connect to the LMTP ports
# !
# TCP|*|225|10.133.152.193|* $Y
# TCP|*|226|10.133.152.193|* $Y

portAccess_optlist = get_mapping("PORT_ACCESS");
#print ("\n -- DEBUG optlist for PORT_ACCESS" . portAccess_optlist .
"\n");

# list of IP addresses
ipaddrs = [feIPs];
push(ipaddrs, myIP);
numips = length(ipaddrs);
ports = ["225", "226"];
numports = length(ports);

i = 1;
loop { #loop over all ipaddrs
  p = numports;
  loop {
    tmp = "TCP|*|" + ports[p] + "|" + ipaddrs[i] + "|*";
    if exists_optlist(get_mapping("PORT_ACCESS"), tmp) {
      warn ( "-- WARNING: optlist for " . tmp . " exists in
PORT_ACCESS");
    } else {
      print("-- INFO: Adding IP " + ipaddrs[i] + " port " + ports[p] + "
to PORT_ACCESS mapping\n");
      prepend_mapping("PORT_ACCESS", [tmp, "$Y"]);
    }
  }
  exitif(p==1);
}

```



```
    p--;
  }
  exitif(i==numips);
  i++;
}
```

```
#####  
\"write -remark backendLMTP.rcp\" to write out the changes\n\";}
```

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 xdfldg=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 `xdfldg=` 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 `xdfldg` is an internal parameter and the bits in it are subject to change or addition without notice. Oracle does not support other clients using this extension with Messaging Server, nor does Oracle support using the Messaging Server LMTP 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, and 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 13. Mail Filtering and Access Control in Unified Configuration

Mail Filtering and Access Control in Unified Configuration

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](#) 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](#) 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 queuing 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 a mappings table is modified, you must recompile the configuration. See [Compiling the MTA Configuration](#).

Access Control Mapping Tables

Mapping Table	Description
<code>SEND_ACCESS</code>	Used to block incoming connections based on envelope <code>From</code> address, envelope <code>To</code> address, source and destination channels. The <code>To</code> address is checked after rewriting, alias expansion, and so on, have been performed.
<code>ORIG_SEND_ACCESS</code>	Used to block incoming connections based on envelope <code>From</code> address, envelope <code>To</code> address, source and destination channels. The <i>original</i> <code>To</code> address after rewriting but before alias expansion is used, but the function of alias expansion has already been performed.
<code>MAIL_ACCESS</code>	Used to block incoming connections based on combined information found in <code>SEND_ACCESS</code> and <code>PORT_ACCESS</code> tables: that is, the channel and address information found in <code>SEND_ACCESS</code> combined with the IP address and port number information found in <code>PORT_ACCESS</code> .
<code>ORIG_MAIL_ACCESS</code>	Used to block incoming connections based on combined information found in <code>ORIG_SEND_ACCESS</code> and <code>PORT_ACCESS</code> tables: that is, the channel and address information found in <code>ORIG_SEND_ACCESS</code> combined with the IP address and port number information found in <code>PORT_ACCESS</code> .
<code>FROM_ACCESS</code>	Used to filter mail based on envelope <code>From</code> addresses. Use this table if the <code>To</code> address is irrelevant.
<code>PORT_ACCESS</code>	Used to block incoming connections based on IP address and TCP port.
<code>IP_ACCESS.</code>	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:

- [Input vs Output Flags](#)
- [Output Flag Argument Order](#)
- [Send Access and Mail Access Mapping Tables](#)
- [SEND_ACCESS and ORIG_SEND_ACCESS Mapping Tables](#)
- [MAIL_ACCESS and ORIG_MAIL_ACCESS Mapping Tables](#)
- [FROM_ACCESS Mapping Table](#)
- [PORT_ACCESS Mapping Table](#)
- [IP_ACCESS Mapping Table](#)

[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 option 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 <i>integer</i>	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 <i>user identifier</i>	Check specified user for specified groupid (UNIX) and if not in the group, reject access (effectively sets the \$N output flag).
\$< <i>string</i>	+++ Send <i>string</i> to UNIX syslog (<i>user.notice</i> facility and severity) if probe matches. (see SNDOPR_PRIORITY option)
\$> <i>string</i>	+++ Send <i>string</i> to UNIX syslog (<i>user.notice</i> facility and severity) if access is rejected. (see SNDOPR_PRIORITY option)

<code>\$Ddelay</code>	Delay response for an interval of <i>delay</i> 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 <code>FROM_ACCESS</code> table; SMTP RCPT TO: command for the other tables).
<code>\$Ttag</code>	Prefix subsequent <i>*ACCESS mapping table probes with tag_tag</i>
<code>\$Aheader</code>	Add the header line <i>header</i> to the message.
<code>\$G conversion_tag</code>	If used in <code>ORIG_SEND_ACCESS</code> , <code>SEND_ACCESS</code> , <code>ORIG_MAIL_ACCESS</code> , and <code>MAIL_ACCESS</code> , 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 <code>FROM_ACCESS</code> , conversion tags are applied to all recipients. <code>\$G</code> is positioned after <code>\$A</code> (header address) in the sequence of arguments read from the mappings. See Mail Conversion Tags . This feature was introduced in Messaging Server 6.0 .
<code>\$Maddress</code>	Capture a copy of the message, sending the captured copy to <i>address</i> . By default, this captured copy is DSN encapsulated---but see the <code>no-</code> argument, <code>non-FROM_ACCESS</code> <code>\$J</code> output flag above.
<code>\$, spamadjust_arg</code>	Set a spam level value <i>x</i> (between -10000.0 and 10000.0). If the message already had a spam level, this is a <code>spamadjust</code> effect (adds or subtracts the specified amount <i>x</i> from the prior spam level). Note that such a spam level/ <code>spamadjust</code> effect applies to all recipients (even for the recipient-specific mapping tables such as <code>SEND_ACCESS</code>) in order for tests to see if one of the recipients is a "honeypot" address. Allows you to perform a sieve <code>spamadjust</code> operation from the access mapping tables. Argument takes the same form as a <code>spamadjust</code> argument. Note also that some of these mappings are applied on a per-recipient basis. Any <code>spamadjust</code> operation that is done applies to all recipients. This feature was introduced in Messaging Server 6.1 .
<code>+\$Ename value</code>	Set the environment variable <i>name</i> to <i>value</i> . This feature was introduced in Messaging Server 7 Update 2 .
<code>\$Xerror-code</code>	Lets you specify the extended SMTP <i>error-code</i> (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: <code>ORIG_SEND_ACCESS</code> <...probe...> \$N\$X4.5.9 Temporary\$ problem\$ with\$ address;\$ try\$ later\$
<code>\$Nstring \$Fstring</code>	Reject access with the optional error text <i>string</i> . <code>\$F</code> and <code>\$N</code> are synonyms.
<code>\$Surl</code>	Apply the Sieve filter obtained from resolving <i>url</i> .
<code>+\$Rn string</code>	Opt in to spam filtering. Only applied if neither <code>\$N</code> nor <code>\$F</code> is set. <i>n</i> is which spam/virus filter package; <i>string</i> 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 queuing 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](#) 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_address
```

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
<code>\$: </code>	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 .
<code>\$:A</code>	Match only if SASL (SMTP AUTH) has been used
<code>\$:E</code>	Match only if ESMTP has been used. This feature was introduced in Messaging Server 6.3 .
<code>\$:L</code>	Match only if LMTP has been used. This feature was introduced in Messaging Server 6.3 .
<code>\$:P</code>	Match only if POP-before-SMTP was used. This feature was introduced in Messaging Server 7 .
<code>\$:T</code>	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 .
\$O	Forces single-copy-per-recipient message copy "split up", as if the single channel option 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
\$Uinteger	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-name_. 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 .
\$Jaddress	Replace original envelope From: address with specified address.

<code>\$Kaddress</code>	Replace the MTA's internal sender address with specified <i>address</i> for subsequent checking. Note: to also add a Sender: header, the source channel must include the <code>authrewrite</code> channel option.
<code>\$Iuser identifier</code>	Check specified user for specified groupid (UNIX) and if not in the group, reject access (effectively sets the \$N output flag).
<code>\$<string</code>	+++ Send <i>string</i> to UNIX syslog (<code>user.notice</code> facility and severity) if probe matches. (see <code>SNDOPR_PRIORITY</code> option)
<code>\$>string</code>	+++ Send <i>string</i> to UNIX syslog (<code>user.notice</code> facility and severity) if access is rejected. (see <code>SNDOPR_PRIORITY</code> option)
<code>\$Ddelay</code>	Delay response for an interval of <i>delay</i> 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 <code>FROM_ACCESS</code> table; SMTP RCPT TO: command for the other tables).
<code>\$Ttag</code>	Prefix subsequent <code>*_ACCESS</code> mapping table probes with tag <i>tag</i>
<code>\$Aheader</code>	Add the header line <i>header</i> to the message.
<code>\$Gconversion_tag</code>	If used in <code>ORIG_SEND_ACCESS</code> , <code>SEND_ACCESS</code> , <code>ORIG_MAIL_ACCESS</code> , and <code>MAIL_ACCESS</code> , 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 <code>FROM_ACCESS</code> , conversion tags are applied to all recipients. <code>\$G</code> is positioned after <code>\$A</code> (header address) in the sequence of arguments read from the mappings. See Mail Conversion Tags . This feature was introduced in Messaging Server 6.0 .
<code>\$Maddress</code>	Capture a copy of the message, sending the captured copy to <i>address</i> . By default, this captured copy is DSN encapsulated---but see the no-argument, non- <code>FROM_ACCESS</code> <code>\$J</code> flag above.
<code>\$Sx,y,z</code>	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 Message Size Limits .
<code>\$, spamadjust_arg</code>	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 <code>SEND_ACCESS</code>) in order for tests to see if one of the recipients is a "honeypot" address. Allows you to perform a sieve <code>spamadjust</code> 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 .
<code>\$(postmaster-address</code>	Set an override postmaster address. This feature was introduced in Messaging Server 6.3 .
<code>) postmaster-address</code>	Set an override postmaster address if none was previously set. This feature was introduced in Messaging Server 6.3 .
<code>\$+Ename value</code>	Set the environment variable <i>name</i> to <i>value</i> . This feature was introduced in Messaging Server 7 Update 2 .

<code>+\$Rn string</code>	Opt in to spam filtering. <i>n</i> specifies which spam/virus filter package; <i>string</i> is the optin string to pass to that spam filter.
<code>\$\$error-code</code>	Lets you specify the extended SMTP <i>error-code</i> (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\$
<code>\$\$Nstring</code> <code>\$\$Fstring</code>	Reject access with the optional error text <i>string</i> . \$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 option (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` channel option 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](#) 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
<code>\$:A</code>	Match only when the probe is performed by the Dispatcher
<code>\$:S</code>	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
<code>\$U</code>	S	Enable channel (slave) debugging . (This feature was introduced in Messaging Server 6.3.); if <code>MM_DEBUG</code> or <code>OS_DEBUG</code> are set, enable those as well. (This feature was introduced in Messaging Server 7u3.)
<code>\$V</code>	S	Enable the MTA's private SMTP extensions XADR, XCIR, XGEN, and XSTA, overriding any SMTP server <code>DISABLE_*</code> channel options. This feature was introduced in Messaging Server 7.
<code>\$/</code>	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 <code>SO_LINGER</code> enabled and a timeout of 0, which may clear slots quicker on intermediate firewalls and proxies. This feature was introduced in Messaging Server 7.
<code>\$Y</code>	DSL	Allow access.
<code>\$T</code>	D	If bit 1 (value 2) of the <code>LOG_CONNECTION</code> MTA option is set and the <code>\$N</code> flag is set so that the connection is rejected, then <code>\$T</code> outputs the entire right hand side text in a "T" (or "X"++) record. The T log entry will include the entire mapping result string (<code>\$N</code> and its string). In contrast, bit 4 of <code>LOG_CONNECTION</code> 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
<code>\$<string</code>	DSL	Send <i>string</i> to syslog if probe matches.
<code>\$>string</code>	DSL	Send <i>string</i> to syslog if access is rejected.
<code>\$Nstring</code> <code>\$Fstring</code>	DSL	Reject access with the optional error text <i>string</i> . <code>\$F</code> is a synonym for <code>\$N</code>
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 <code>LOG_CONNECTION</code> MTA option is set to bit 4 (value 16), <code>PORT_ACCESS</code> 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.
<code>\$Ddelay</code>	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 <code>BANNER_PURGE_DELAY</code> value. Note that any <code>PORT_ACCESS</code> mapping setting overrides the <code>BANNER_PURGE_DELAY</code> SMTP channel option. See Anti-Spam Technique: Delay Sending the SMTP Banner for details on using this anti-spam feature.
<code>\$Schannel</code>	S	Set <i>channel</i> 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 `msconfig` options of the form `authoption-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 `base.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||xyzy
* $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 `xyzy` 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|||xyzy
* $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` channel option effect (see [Specify a last resort host for delivery](#)).

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.

X

IP_ACCESS Mapping Table Flags

The following flags apply to the `IP_ACCESS` mapping table. See [Mappings](#) 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` channel options are present on the source channel. (See [SMTP authentication and SASL](#).)

`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 `imsimta test` in *Messaging Server Administration Reference* for more information. The following example 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 `$/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/significant-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` option on your inbound SMTP channel, usually the `tcp_local` channel, and is set by default.

The `switchchannel` channel option 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` option:

```
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` options; 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` option; 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` option. The option is applied to the `tcp_local` channel. When a message comes in your SMTP server, that option 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` option, 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` option on your inbound SMTP channel, usually the `tcp_local` channel.

The `saslswitchchannel` option 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 mustsaslsrvr 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 `mustsaslsrvr` on it.

2. Modify your `tcp_local` channel by adding `maysaslsrvr` and `saslswitchchannel tcp_auth`, as shown in the following example:

```
tcp_local smtp mx single_sys maysaslsrvr 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 option 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` option, 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 configuration. For example, for the `smtp` service, set instances of the option to the various lists you want to check against:

```

msconfig
msconfig# set dispatcher.service:smtp.dns_verify_domain
sbl-xbl.spamhaus.org
msconfig# set dispatcher.service:smtp.dns_verify_domain list.dsbl.org
msconfig# write

```

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 MTA `LOG_CONNECTION` option 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

STOPPED HERE

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 Option URL-pattern Substitution Tags (Case-insensitive)

Tag	Meaning
*	Perform group expansion.
**	Expand the attribute <code>mailForwardingAddress</code> . 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 \$\$, but if no subaddress is available just insert nothing
\$2S	As \$\$, but if no subaddress is available insert nothing and delete the preceding character
\$3S	As \$\$, 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, <code>LDAP_SPARE_1</code>
\$F	Insert the name of the delivery file (<code>mailDeliveryFileURL</code> attribute)
\$G	Insert the value of the second spare attribute, <code>LDAP_SPARE_2</code>
\$H	Substitute in host
\$I	Insert the hosted domain (part of UID to the right of the separator specified by <code>domainUIdSeparator</code>). 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 (<code>mailProgramDeliveryInfo</code> 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 <code>{+}</code> , but can be specified by the MTA option <code>SUBADDRESS_CHAR</code> . 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, <code>{+}</code> .

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 option enables message filtering on messages enqueued *to* the channel to which it is applied. The `sourcefilter` channel option enables message filtering on messages enqueued *by* (from) the channel to which it is applied. These options 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 option is:

```
destinationfilter URL-pattern
```

The syntax for the `sourcefilter` channel option 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 option enables message filtering on the channels to which it is applied. The option 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 option 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 Option URL-pattern Substitution Tags \(Case-insensitive\)](#).

The `fileinto` option 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 option 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 MTA option `FILTER_DISCARD` to 2 by running `msconfig set filter_discard 2`. 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, run `msconfig edit channels` and check that the `ims-ms` channel 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 user'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` option in [Master_debug, nomaster_debug, slave_debug, noslave_debug Channel Options](#).

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 14. Managing Logging in Unified Configuration

Managing Logging in Unified Configuration

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](#)
- [Managing MTA Message and Connection Logs](#)
- [Managing Message Store, Admin, and Default Service Logs](#)
- [Using Message Store Log Messages](#)
- [MMP Logging](#)

Overview of Logging

This section contains the following subsections:

- [What Is Logging and How Do You Use it?](#)
- [Types of Logging Data](#)
- [Types of Messaging Server Log Files](#)
- [Tools for Managing Logging](#)
- [Tracking a Message Across the Various Log Files](#)

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, logging provides you with two types of information:

- Operational data
- Error conditions, also known as event logging

For the most part, Messaging Server logging provides operational data. This operational data contains information 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; and at a later point in time, when the message was removed from disk and inserted into user's mailbox.

However, Messaging Server logging does also provide some event logging data. To obtain event logging data, 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. Channel options <code>master_debug</code> and <code>slave_debug</code> turn on channel debugging, which increases the verbosity of the channel log files. Level and type of information is controlled with the various <code>*_DEBUG</code> MTA options.	<i>channel-name_master.log*</i> (example: <code>tcp_local_master.log*</code>) <i>channel-name_slave.log*</i> (example: <code>tcp_local_slave.log*</code>)
IMAP	Contains logged events related to IMAP4 activity of this server	imap, <code>imap.sequenceNum.timeStamp</code>
POP	Contains logged events related to POP3 activity of this server	pop, <code>pop.sequenceNum.timeStamp</code>
HTTP	Contains logged events related to HTTP activity of this server	http, <code>http.sequenceNum.timeStamp</code>
Default	Contains logged events related to other activity of this server, such as command-line utilities and other processes	default, <code>default.sequenceNum.timeStamp</code>
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 `msconfig` command.

For 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 you configure MTA logging by specifying 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 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.

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](#)

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.

See [Enabling MTA Logging](#) for more information.

Enabling logging causes the MTA to write 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 as follows:
`<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.
5. 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` MTA option. The SMS channel can be configured to log a page count rather than file size in this field. See [LOG_PAGE_COUNT](#).
6. 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.
7. The original form of the envelope `To:` address (in the example, `marlowe@siroe.com`).
8. The active (current) form of the envelope `To:` address (in the example, `marlowe@ims-ms-daemon`).
9. 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: <code>imsimta qm "delete"</code> command always generates a "K" record for each recipient, while a <code>qm "return"</code> 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	LMTMP deposit into the message store. This action code is used on the LMTMP 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 <code>EEC</code> or <code>DEC</code> .
E	An EHLO command was issued/accepted and therefore ESMTP was used.
L	LMTMP 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 <code>connection.log_current</code> (or <code>{mail.log_current</code> 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 MTA options all enabled, 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
<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 enqueueing 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 options; see [IDENT Lookups](#) for an instance of the default `identnone` option, 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 option 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. Run the `msconfig edit channels` command.
2. To enable logging for a particular channel, add the `logging` option to the channel definition. For example:

```
channel-name option1 option2 logging
```

In addition, you can also set a number of configuration parameters such as directory path for log files, log levels, and 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. Run the `msconfig edit channels` command.
2. Add the `logging` option to your `defaults` channel configuration file (see [Configuring Channel Defaults](#)). For example:

```
defaults notices 1 2 4 7 copywarnpost copysendpost postheadonly
noswitchchannel \
immonurgent maxjobs 7 defaulthost siroe.com siroe.com logging

!
! delivery channel to local /var/mail store
1 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.

To Send MTA Logs to syslog

1. Enter the following command to set `log_messages_syslog` to 1:

```
msconfig set log_messages_syslog 1
```

To write MTA message log file entries to syslog, you need to set the `log_messages_syslog` option to a non-zero value. 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. 0 is the default and means no syslog or event logging is performed.

Facility and priority numbers are located in the `/usr/include/sys/syslog.h` file.

To Control Formatting of Log Entries

1. Set the `LOG_FORMAT` option by running `msconfig set log_format n` where `n` corresponds to one of the settings below
 - 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="1" 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$7c1clee0$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="0" 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

- Set the `LOG_MESSAGE_ID` option to 1 by running `msconfig set log_message_id 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

- Set the `LOG_QUEUE_TIME` option to 1 by running `msconfig set log_queue_time 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

- Set the `LOG_FILENAME` option to 1 by running `msconfig set log_filename 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

- Set the `LOG_CONNECTION` option by running `msconfig set log_connection 1`
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

- Set the `SEPARATE_CONNECTION_LOG` option to 1 by running `msconfig set separate_connection_log 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

- Set the `LOG_PROCESS` option by running `msconfig set log_process 1`.
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

- Set the `LOG_USERNAME` option by running `msconfig set log_username 1`.
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 following example shows a 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 <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)
<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)
<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 <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
<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
<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
<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
<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
<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 <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
<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 <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
<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
<0JDQ00904RSK9Z00@sesta.com>,<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
<0JDQ00906RSK9Z00@sesta.com>,<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
<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
<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
<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
<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
<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
<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
<0JE100J09UUB4N00@sesta.com>,<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
<0JE100J0BUUB4N00@sesta.com>,<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
<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
<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
<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
<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
<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
<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)
<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)
<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)
<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
<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 <user@some.org >... Recipient ok

```

1. The message comes in the `tcp_intranet` 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 option. 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)
<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)
<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)
<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
<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
<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
<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 <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 O 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.
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. A default dispatcher configuration is created at installation time and can be used without any changes made. However, if you want to modify the default configuration for security or performance reasons, you can do so by running the `msconfig` command.

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. Run the `msconfig set dispatcher.debug -1`. (See [Restricted Options](#).)
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 affect 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 `cs` 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:

- [msconfig Logging Options](#)
- [Understanding Service Log Characteristics](#)
 - [Logging Levels](#)
 - [Categories of Logged Events](#)
 - [Service Log File Directories](#)
- [Understanding Service Log File Format](#)
 - [Store and Administration Log File Components](#)
- [Defining and Setting Service Logging Options](#)
 - [Flexible Logging Architecture](#)
 - [Planning the Options You Want](#)
 - [Understanding Logging Options](#)
- [Searching and Viewing Service Logs](#)
 - [Search Parameters](#)
- [Working With Service Logs](#)
 - [To Send Service Logs to syslog](#)
 - [To Disable HTTP Logging](#)
 - [To Set the Server Log Level](#)
 - [To Specify a Directory Path for Server Log Files](#)
 - [To Specify a Maximum File Size for Each Service Log](#)
 - [To Specify a Service Log Rotation Schedule](#)
 - [To Specify a Maximum Number of Service Log Files Per Directory](#)
 - [To Specify a Storage Limit](#)
 - [To Specify the Minimum Amount of Free Disk Space to Reserve](#)
 - [To Specify an Age for Logs at Which They Expire](#)
- [Implementing and Configuring Message Store Transaction Logging](#)
 - [Overview of Message Store Transaction Logging](#)
 - [Message Store Transaction Logging Log Entries](#)

- Configuring Message Store Transaction Logging
- Message Store Transaction Log Examples
- Other Message Store Logging Features
- Message Store Logging Examples
 - Message Store Logging Example: Bad Password
 - Message Store Logging Example: Account Disabled
 - Message Store Logging Example: Message Appended
 - Message Store Logging Example: Message Retrieved by a Client
 - Message Store Logging Example: Message Removed from a Folder
 - Message Store Logging Example: Duplicate Login Messages

msconfig Logging Options

To control the location of log files, use the following options for specifying directory paths:



Note

The location of MTA log files, which are in the `msg-install-path/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 `msconfig` options to specify non-default locations for non-MTA log files.

msconfig Directory Paths for Log Files

Parameter	Description
*.logfile.logdir base.logfile.logdir dispatcher.logfile.logdir ens.logfile.logdir http.logfile.logdir imap.logfile.logdir imaproxy.logfile.logdir mta.logfile.logdir job_controller.logfile.logdir metermaid.logfile.logdir mmp.logfile.logdir msadmin.logfile.logdir messagetrace.logfile.logdir pop.logfile.logdir popproxy.logfile.logdir snmp.logfile.logdir submitproxy.logfile.logdir tcp_lmtp_server.logfile.logdir	Directory path for log files. If this is not specified, log files will be placed in the <code>msg-install-path/data/log</code> directory. For the MTA, this option is only used by Message Store insertion tasks Directory path to the <code>imta</code> log file used for Message Store insertion (<code>ims_master</code> , LMTP). It is not used by other parts of the MTA which always log to the default location. The default location is <code>msg-install-path/data/log</code> . Changing that path to a soft-link is supported. (Restart of all services required). Syntax: <code>dirpath</code>

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 `service.logfile.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
<i>dateTime</i>	The date and time at which the event was logged, expressed in <i>dd/mm/yyyy hh:mm:ss</i> format, with a time-zone field expressed as <i>+/-hhmm</i> from GMT. For example: 02/Jan/1999:13:08:21 -0700
<i>hostName</i>	The name of the host machine on which the server is running: for example, <i>showshoe</i> . Note: If there is more than one instance of Messaging Server on the host, you can use the process ID (<i>pid</i>) to separate logged events of one instance from another.
<i>processName</i>	The name of the process that generated the event: for example, <i>cgi_store</i> .
<i>pid</i>	The process ID of the process that generated the event: for example, <i>18753</i> .
<i>category</i>	The category that the event belongs to: for example, <i>General</i> (see Example).
<i>logLevel</i>	The level of logging that the event represents: for example, <i>Notice</i> (see Example).
<i>eventMessage</i>	An event-specific explanatory message that may be of any length: for example, <i>Log created (894305624)</i> .

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 `msconfig` command with the `syslogfacility` option:

```
msconfig logfile.service.syslogfacility 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:

```
msconfig  
msconfig> set http.enable 0  
msconfig# set http.enablesslport 0  
msconfig# write
```

To Set the Server Log Level

1. Run the following `msconfig` command:

```
msconfig set <service>.logfile.loglevel <loglevel>
```

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 `msconfig` command:

```
msconfig <service>.logfile.logdir -v <dirpath>
```

To Specify a Maximum File Size for Each Service Log

1. Run the following `msconfig` command:

```
msconfig set <service>.logfile.maxlogfilesize <size>
```

where `<size>` specifies a number of bytes.

To Specify a Service Log Rotation Schedule

1. Run the following `msconfig` command:

```
msconfig set <service>.logfile.rollovertime <number>
```

where `number` specifies a number of seconds.

To Specify a Maximum Number of Service Log Files Per Directory

1. Run the following `msconfig` command:

```
msconfig <service>.logfile.maxlogfiles <number>
```

where `number` specifies the maximum number of log files.

To Specify a Storage Limit

1. Run the following `msconfig` command:

```
msconfig <service>.logfile.maxlogsize <number>
```

where `number` specifies a number in bytes.

To Specify the Minimum Amount of Free Disk Space to Reserve

1. Run the following `msconfig` command:

```
msconfig <service>.logfile.minfreediskspace <number>
```

where `number` specifies a number in bytes.

To Specify an Age for Logs at Which They Expire

```
msconfig -o <service>.logfile.expirytime <number>
```

where *number* specifies a number in seconds.

Implementing and Configuring Message Store Transaction Logging

This section contains the following topics:

- [Overview of Message Store Transaction Logging](#)
- [Message Store Transaction Logging Log Entries](#)
- [Configuring Message Store Transaction Logging](#)
- [Message Store Transaction Log Examples](#)

Overview of Message Store Transaction Logging

You can use Message Store transaction logging to record Messaging Server user actions and events, tracing messages similar to the way the MTA traces messages. Tracing messages in this fashion allows you to track critical events of a message's life cycle.

Message Store transaction logging uses the same XML format used in the MTA `mail.log*` and `connection.log*` files. This XML format is the default. It provides the same transaction information previously logged in the process log at the `notice` and `information` log levels as well as information in `msgtrace` logging. It also includes IMAP and POP context numbers in all IMAP and POP log messages. You do not set log levels for Message Store transaction logs. You instead configure settings to determine which events and attributes to log. Message Store transaction logging uses a log roll over daemon which rolls over all logs previously written using `nslogger`. Message Store transaction logging provides a backward-compatibility mode that behaves the way the previous store logging worked, combining process and action logging together. It provides a `msgtrace` mode also for backward compatibility.

Message Store Transaction Logging Log Entries

Message Store transition logging supports the following user actions or events.

```
li - login action
ma - append message action
ex - expunge message action
fe - fetch or retrieve message action
qc - quota change action
qe - quota exceeded limit action
mc - create mailbox action
md - delete mailbox action
mr - rename mailbox action
ms - subscribe to mailbox action
mu - unsubscribe to mailbox action
ac - set mailbox access control list action
sl: select a mailbox
co: connect to a mail account
```

Message Store transaction logging actions can support the following attributes.

```
ts - time stamp
pi - process id
si - session id
us - canonical user name, post authentication canonical user name
mi - message id
ii - folder id
tr - transport information
if - imap flags
fi - file name
sz - message size
no - node name
tg - RFC 5423 tag
ac - action flags
ct - time cost
ma - mailbox id, mailbox, URI encoded, plus uidvalidity
om - old mailbox id
ut - RFC 5423 uidset
ua - sasl authentication id, RFC 5423 "admin"
bs - body structure, RFC 5423
dq - disk quota, RFC 5423
du - disk used
qt - over quota trigger
qr - rule with over quota
ev - message envelope
mm - maximum messages
mc - message count
sq - module sequence
sn - service name, imap, pop, admincli
un - next uid
uv - uidvalidity
ui - uid
cx - context number
ur - IMAP URL
uo - SASL original user identity
at - SSL authorization type
cs - SSL cipher suite if SSL is used
cn - SSL server certificate nickname
cc - SSL client certificate subject
su - mailbox specialuse flag
ai - setacl identifier
ar - setacl rights
hl - message header size
sd - number of deleted messages
mq - message quota
mu - module sequence in milliseconds
uc - unchanged since in seconds
uu - unchanged since in milliseconds
lu - old uid
ls - old uidset
nt - anything worthy noting
```



Note

Not all attributes are available to each action. The following attributes are available for all actions:

```
ts: time stamp,  
pi: process id,  
sn: service name, imap, pop, admincli
```

An action log entry is an XML element with the following characteristics:

- The element tag is the two character action label, which follows XML syntax to start and end: <tag ... />
- The element has many attributes defined previously, and each action has own attributes.
- The attribute starts with a two character attribute label, and its value is quoted.
- The following attributes are required for each action entry: timestamp (*ts*), service name (*sn*), and process id (*pi*).

A typical log entry resembles the following:

```
<co ts="09/Feb/2015:13:44:54.059 -0800" sn="imapd" pi="7149" ac="C"  
tr="[127.0.0.1:49487]" us="admin" nt="2015/2/9 13:43:52 0:01:02 158 1032  
1" />
```

where:

- *co* indicates the action label, which is connection closed.
- Three mandatory attributes, *ts* (time stamp), *sn* (service name), and *pi* (process id) are present.
- The action specific attributes are *us* (user name) and *ac* (action flags).

Configuring Message Store Transaction Logging

Transaction logging configuration is applied globally to IMAP, POP, and delivery Message Store components.

Enabling Message Store Transaction Logging

To trace transactions in the Message Store transaction log, you configure message tracing in addition to the logging configuration.

To enable Message Store transaction logging, run the following command:

```
msconfig set messagetrace.activate transactlog
```

To show the configuration, run the following command:

```
msconfig show messagetrace.activate  
role.messagetrace.activate = transactlog
```

Enabling Message Store Transaction Log Actions and Attributes

The syntax to enable store actions and attributes is a matrix that you configure separately using actions and action attributes.

The following table shows the actions and their descriptions.

Action	Description
all	All actions are enabled. This is the default.
+({li lo ma ...})	Only actions in the list are enabled. Actions are separated by a space.
-({li lo ma ...})	All actions are enabled except those in the list. Actions are separated by a space.

To enable all Message Store actions except for login and logout actions, run the following command.

```
msconfig set imap.actions '-(li lo)'
```

Running the `msconfig show imap.actions` command shows the `imap.actions` settings.

```
msconfig show imap.actions
role.imap.actions = -(li lo)
```

The following table shows the attributes and their descriptions.

Attributes	Description
all	All attributes are enabled. This is the default.
+({us mi ii ...})	Only attributes in the list are enabled. Attributes are separated by a space.
-({us mi ii ...})	All attributes are enabled except those in the list. Attributes are separated by a space.

To enable only the canonical user name and session ID attributes, run the following command.

```
msconfig set imap.actionattributes '+(us si)'
```

Running the `msconfig show imap.actionattributes` shows the `imap.actionattributes` settings.

```
msconfig show imap.actionattributes
role.imap.actionattributes = +(us si)
```

Message Store Transaction Log Examples

The examples in this section use the following Messaging Store transaction logging settings:

```
msconfig show messagetrace
role.messagetrace.activate = transactlog

msconfig show imap.actions
role.imap.actions = all

msconfig show imap.actionattributes
role.imap.actionattributes = all
```

Example Message Store Transaction Logs for IMAP

Example 1: Log in as Joe.

```
<li ts="24/Jul/2014:13:33:04.306 -0700" sn="imapd" pi="26435" us="joe"
tr="[127.0.0.1:45046]" at="plaintext" cs="noSSL" si="0"/>
```

Example 2: Create mailbox Folder1 and Folder2.

```
<mc ts="24/Jul/2014:13:33:40.902 -0700" sn="imapd" pi="26435" us="joe"
ma="Folder1"/>
<mc ts="24/Jul/2014:13:34:05.299 -0700" sn="imapd" pi="26435" us="joe"
ma="Folder2"/>
```

Example 3: Delete mailbox Folder1.

```
<md ts="24/Jul/2014:13:34:12.515 -0700" sn="imapd" pi="26435" us="joe"
ma="Folder1"/>
```

Example 4: Append message to Folder2.

```
<ma ts="24/Jul/2014:13:46:25.271 -0700" sn="imapd" pi="27080" us=""
ma="user/joe/Folder2" sz="163" ui="1" uv="1406234045" cx="0"/>
```

Example 5: Expunge 2 messages in the Inbox.

```
<ex ts="25/Jul/2014:13:50:27.164 -0700" sn="imapd" pi="30247"
ma="user/joe" mi="0N8S00A0B007SS00@host.example.com"
no="[127.0.0.1:52383]"/>
<ex ts="25/Jul/2014:13:52:43.163 -0700" sn="imapd" pi="30247"
ma="user/joe" mi="0N8R00A06ZLSS00@host.example.com"
no="[127.0.0.1:52383]"/>
```

Example 6: Log out.

```
<co ts="09/Feb/2015:13:44:54.059 -0800" sn="imapd" pi="7149" ac="C"
tr="[127.0.0.1:49487]" us="admin" nt="2015/2/9 13:43:52 0:01:02 158 1032
1"/>
```

Example 7: Connect to an email account.

```
<co ts="05/Feb/2015:11:57:41.896 -0800" sn="imapd" pi="27433"
tr="[192.0.2.1:3984]" at="ssl"/>
```

Example 8: Select a folder.

```
<sl ts="05/Feb/2015:11:57:42.005 \-0800" sn="imapd" pi="27433"
tr="[192.0.2.1:3984]" ma="user/admin" us="admin"/>
```

Example 9: Rename a folder.

```
imap command: A01 RENAME Folder1 Folder2
<mr ts="05/Feb/2015:12:03:52.672 \-0800" sn="imapd" pi="27433"
us="admin" ma="Folder2" om="Folder1"/>
```

Example 10: Subscribe to a folder.

```
imap command: A01 SUBSCRIBE INBOX
<ms ts="05/Feb/2015:12:08:21.589 \-0800" sn="imapd" pi="27433"
us="admin" ma="inbox"/>
```

Example 11: Unsubscribe from a folder.

```
imap command: A01 UNSUBSCRIBE INBOX
<mu ts="05/Feb/2015:12:14:39.187 \-0800" sn="imapd" pi="3613" us="admin"
ma="inbox"/>
```

Example 12: Setacl to a mailbox.

```
imap command: A01 SETACL folder1 david lrstwiead
<ac ts="05/Feb/2015:12:06:34.287 \-0800" sn="imapd" pi="27433"
us="admin" ma="user/admin/folder1" nt="admin lrstwiepkxancd :admin
lrstwiepkxancd David lrstwiead "/>
```

Example 13: Fetch rfc822.text.

```
imap command: A01 FETCH 1 RFC822.TEXT
<fe ts="05/Feb/2015:14:00:54.865 \-0800" sn="imapd" pi="3613" us="admin"
ma="user/admin" sz="-1:0" mi="<54C145D2.7010202@shenmail.com>/>
```

Example 14: Change quota.

```
imap command: A01 SETQUOTA user/admin/Folder1 (STORAGE 512)
<qc ts="05/Feb/2015:14:15:36.059 \-0800" sn="imapd" pi="3613" us="admin"
ur="user/admin/Folder1" dq="512"/>
```

Example 15: Quota exceeds limit.

```
imquotacheck command: ./imquotacheck \-n \-l \-r rulefile
<qe ts="05/Feb/2015:15:22:54.087 \-0800" sn="imquotacheck" pi="26490"
us="admin" dq="2" du="43" mq="0" mc="0" qt="90" qr="rule3"/>
```

For more information on syntax and options, see the `imquotacheck` documentation in *Messaging Server System Administrator's Guide*.

Example Message Store Transaction Logs for POP

Example 1: Log in as Joe.

```
<li ts="24/Jul/2014:16:59:43.187 \-0700" sn="popd" pi="27522" us="joe"
tr="[127.0.0.1:29611]" at="plaintext" cs="noSSL" si="21"/>
```

Example 2: Fetch message.

```
<fe ts="24/Jul/2014:17:00:08.205 \-0700" sn="popd" pi="27522" us="joe"
om="user/joe" mi="<0N8S00A0E09CSS00@host.example.com>" />
```

Example 3: Delete message.

```
<ex ts="24/Jul/2014:17:02:04.565 \-0700" sn="popd" pi="27522"
ma="user/joe" mi="<0N8S00A0E09CSS00@host.example.com>" no="sc11136733"/>
```

Example 4: Connect to an email account.

```
<co ts="05/Feb/2015:12:01:36.719 -0800" sn="popd" pi="27443"
tr="[10.133.141.196:60395]" at="" />
```

Example Message Store Transaction Logs for Message Delivery

Example 1: Send five messages to Joe.

```

<ma ts="24/Jul/2014:17:05:13.950 \-0700" sn="ims_master" pi="9236" us=""
ma="user/joe" sz="404652" mi="<0N9800L0UQWP9600@host.example.com>"
ui="5" uv="1392927327" cx="0"/>
<ma ts="24/Jul/2014:17:05:13.954 \-0700" sn="ims_master" pi="9236" us=""
ma="user/joe" sz="404652" mi="<0N9800L0WQWP9600@host.example.com>"
ui="6" uv="1392927327" cx="0"/>
<ma ts="24/Jul/2014:17:05:13.974 \-0700" sn="ims_master" pi="9236" us=""
ma="user/joe" sz="404652" mi="<0N9800L0YQWP9600@host.example.com>"
ui="7" uv="1392927327" cx="0"/>
<ma ts="24/Jul/2014:17:05:14.010 \-0700" sn="ims_master" pi="9240" us=""
ma="user/joe" sz="404652" mi="<0N9800L10QWP9600@host.example.com>"
ui="8" uv="1392927327" cx="0"/>
<ma ts="24/Jul/2014:17:05:14.043 \-0700" sn="ims_master" pi="9240" us=""
ma="user/joe" sz="404652" mi="<0N9800L12QWQ9600@host.example.com>"
ui="9" uv="1392927327" cx="0"/>

```

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. For more information, see the topic on checking user IMAP/POP/Webmail session by using telemetry in the "Monitoring the Message Store" section in *Messaging Server System Administrator's Guide*.

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 `lmt_p` 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:<Roam.SIMC.2.0.6.1093995286.11265.user1@vadar.siroe.co
```

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:<Roam.SIMC.2.0.6.1093051161.3655.user1@vad.siroe.com
```

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 in *Messaging Server System Administrator's Guide*.** 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.
- **[Logging Parameters](#).** Search for `logfile`. The information lists the configuration parameters that you set for Messaging Server by using the `configutil` command. You can view the documentation for the Unified Configuration equivalent of a `configutil` option by clicking on the `configutil` name link.
- **[General Discussion on Message Store Logging](#).** Describes logging for the Message Store (POP, IMAP, and HTTP), Admin, and Default services.

MMP Logging

See [MMP Reference](#).

Chapter 15. Message Store and Mailbox Management in Unified Configuration

Message Store and Mailbox Management in Unified Configuration

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 the Message Store Command-line Utilities section in *Messaging Server System Administrator's Guide*.

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 in Unified Configuration](#)). 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:

- [Administering Very Large Mailboxes in Unified Configuration](#)
- [Backing Up and Restoring the Message Store in Unified Configuration](#)
- [Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS in Unified Configuration](#)
- [Configuring Message Expiration in Unified Configuration \(Tasks\)](#)
- [Configuring POP, IMAP, and HTTP Services in Unified Configuration](#)
- [Handling Message Store Overload in Unified Configuration](#)
- [Managing Message Store Partitions and Adding Storage in Unified Configuration](#)
- [Managing Message Store Quotas in Unified Configuration](#)
- [Managing Message Types in the Message Store in Unified Configuration](#)
- [Managing Shared Folders in Unified Configuration](#)
- [Message Store Administration in Unified Configuration](#)
- [Message Store Architecture and Concepts in Unified Configuration](#)
- [Message Store Automatic Recovery On Startup in Unified Configuration](#)
- [Message Store Maintenance Queue in Unified Configuration](#)
- [Message Store Message Types Overview in Unified Configuration](#)
- [Message Store Quota \(Overview\) in Unified Configuration](#)
- [Migrating Mailboxes to a New System in Unified Configuration](#)
- [Monitoring Disk Space in Unified Configuration](#)
- [Protecting Mailboxes from Deletion or Renaming \(Unified Configuration\)](#)
- [Reducing Message Store Size Due to Duplicate Storage in Unified Configuration](#)
- [Specifying Administrator Access to the Message Store in Unified Configuration](#)

- [Troubleshooting the Message Store in Unified Configuration](#)
- [Valid Message Store UIDs and Folder Names in Unified Configuration](#)

Administering Very Large Mailboxes in Unified Configuration

Administering Very Large Mailboxes in Unified Configuration

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](#)
- [The Structure of a Mailbox](#)
- [Mailbox Size Limit](#)
- [Mailbox Migration](#)
- [Pre-Deployment Preparations](#)
- [Checking Mailbox Data](#)

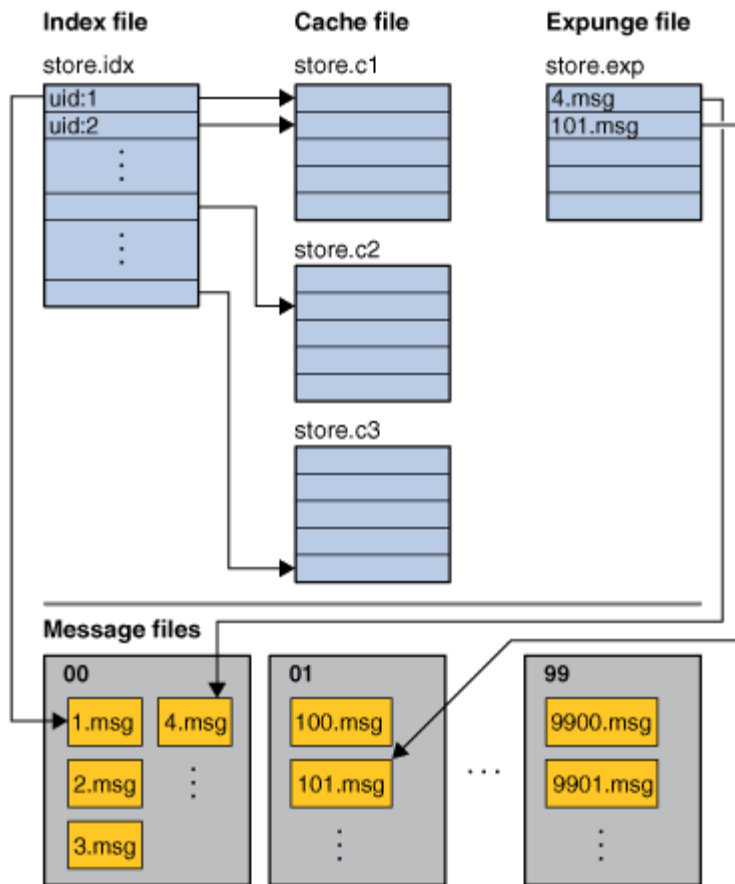
Very Large Mailboxes Overview

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.maxcachefilesizesize`). 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 in Unified Configuration](#). 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 `msconfig` option `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 `store.maxmessages`, quotas or expire rules to prevent mailboxes from getting too big

The new `msconfig` options for managing large mailboxes are:

msconfig Option	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 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 in Unified Configuration

Backing Up and Restoring the Message Store in Unified Configuration

This information describes how to back up and restore mailboxes. For conceptual information on the message store, see the following:

- [Message Store Disaster Backup and Recovery](#)
- In *Messaging Server System Administrator's Guide*, these topics:
 - [Administering Message Store Database Snapshots \(Backups\)](#)
 - [Message Store Directory Layout](#)

Topics:

- [Mailbox Backup and Restore Overview](#)
- [To Create a Mailbox Backup Policy](#)
- [To Create Backup Groups](#)
- [To Run the imbackup Utility](#)
- [To Restore Mailboxes and Messages](#)
- [To Use StorageTek Enterprise Backup Software](#)
- [To Use a Third Party Backup Software \(Besides StorageTek Enterprise Backup Software\)](#)
- [Troubleshooting Backup and Restore Problems](#)
- [Message Store Disaster Backup and Recovery](#)

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 `imbackup` 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. `imbackup` 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.

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 `store.backupexclude` option.

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/sbin
# ./msconfig set store.backupexclude "Trash%Bulk Mail%Third Class Mail"
# ./msconfig show store.backupexclude
role.store.backupexclude = Trash%Bulk Mail%Third Class Mail
```

The backup utility backs up all folders in a user mailbox except those folders specified in the `store.backupexclude` option.

This feature works with the Messaging Server backup utility, StorageTek Enterprise Backup Software, and third-party backup software.

You can override the `store.backupexclude` 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.

To Back Up Data By Using StorageTek Enterprise Backup Software

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" in *Messaging Server System Administrator's Guide*.
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.



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: savenpc;
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 `savenpc` 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: savenpc
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 `imsbackup` 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 `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 `store.snapshotpath` option.)
- Configuration data. Including the local configuration file at `msg-svr-base/data/config`. See also "Message Store Directory Layout" in *Messaging Server System Administrator's Guide*.

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.

Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS in Unified Configuration

Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS in Unified Configuration

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](#)
- [Configuration Recommendations for ZFS and Messaging Server](#)
- [To Configure ZFS and Messaging Server](#)
- [ZFS Administration Recommendations](#)
- [ZFS Reference](#)

Before You Begin

Before using ZFS to back up the Messaging Server message store, read the topic on Messaging Server and tiered storage overview in *Messaging Server System Administrator's Guide*, which describes message store operation, its performance characteristics, and how to plan for and allocate store partitions.

Configuration Recommendations for ZFS and Messaging Server

The basic recommendations for configuring ZFS and Messaging Server are:

1. Separating the Messaging Server `mboxlist` database, message file, and index cache files on different file systems
2. Configuring the index cache record file system to use the `recordsize` of 4 Kbytes
3. Disabling file access time record
4. Keeping ZFS pool space under 80 percent utilization to maintain pool performance

The following information provides more context on these recommendations.

mboxlist Database, Message File and Index Cache Files Overview

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 setting the `partition:partition_name.path` option, 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 setting the `partition:partition_name.messagepath` option, 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.

Index Cache Record File System

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 to 4 Kbytes for these file systems can improve performance and reduce incremental snapshot backup size.

Access Time Record

The message store does not utilize the file access time. By disabling file access time updates, you reduce unnecessary overhead.

ZFS Pool Space Utilization

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 `alarm.system:diskavail.threshold` option to at least 20, to receive a warning before the disk becomes full. (The default value is 10). To enable message throttling sooner, configure the `store.diskusagethreshold` to 90.

To Configure ZFS and Messaging Server

The following steps implement the previously discussed recommendations.

1. Separate the Messaging Server `mboxlist` database, message file, and index cache files on different file systems.

For example:

```
# zfs create store/mboxlist
# zfs set mountpoint=/var/opt/sun/comms/messaging/store/mboxlist
store/mboxlist
# zfs create store/primary-idx
# msconfig set partition:primary.path /store/primary-idx
# zfs create store/primary-msg
# msconfig set partition:primary.messagepath /store/primary-msg
```

2. Configure the index cache record file system to use the recordsize of 4 Kbytes.

For example:

```
# zfs set recordsize=8k store/primary-idx
# zfs set recordsize=128k store/primary-msg
# zfs set recordsize=8k store/mboxlist
```

Notes

- This setting controls the recordsize of newly created files only.
- Do not set recordsize smaller than the system page size (8 Kbytes on SPARC and 4 Kbytes on Intel).
- 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.

For example:

```
# zfs set atime=off store/mboxlist
# zfs set atime=off store/primary-idx
# zfs set atime=off store/primary-msg
```

4. Configure the disk usage alarm threshold `alarm.system:diskavail.threshold` option to at least 20, to receive a warning before the disk becomes full. (The default value is 10).

For example:

```
# msconfig set alarm.system:diskavail.threshold 20
```

5. To enable message throttling sooner, configure the `store.diskusagethreshold` option to 90. (The default is 99).

For example:

```
# msconfig set store.diskusagethreshold 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. For example:

```
# 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
```

- Perform incremental backups.
You can use `zfs send -i` to perform incremental backups. Destroy the snapshots when they are not needed. For example:

```
# zfs destroy -r store@now
```

**Note**

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.

ZFS Reference

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

http://www.solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide

Configuring Message Expiration in Unified Configuration (Tasks)

Configuring Message Expiration in Unified Configuration (Tasks)

This information describes the tasks you use to expire messages. See the topic on message store message expiration in *Messaging Server System Administrator's Guide* 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](#)

To Set imexpire Rules Textually



Note

In Unified Configuration, you can continue to configure messaging expiration as explained in this information, or use the `msconfig` command to individually set the appropriate configuration options. Modifying the `store.expirerule` file enables access to more functionality than using the `msconfig` command.

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 the following 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 Expiration Rules by Using the `msconfig` Command

- In Unified Configuration, you can set expiration rules by using the following `msconfig` command:

```
./msconfig set expirerule:<name>.<option> <value>
```

For example:

```
./msconfig set expirerule:Rule1.folderpattern user/. *
./msconfig set expirerule:Rule1.messagesize 100000
./msconfig set expirerule:Rule1.messagesizedays 3
```

You can set the following options in this way:

imexpire Unified Configuration Options

Option
deleted
exclusive
folderpattern
foldersizebytes
messagecount
messagedays
messagesize
messagesizedays
seen

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 <i>userid</i> .	<code>user/userid/*</code>	<code>user/userid/.*</code>
Apply rule to messages of <i>userid</i> in folder <code>Sent</code> .	<code>user/userid/Sent</code>	<code>user/userid/Sent</code>
Apply rule to entire message store.	<code>user/*</code>	<code>user/.*</code>
Apply rule to any folder called <code>Trash</code> anywhere in any user's hierarchy.	<code>user/*/Trash</code>	<code>user/.*/Trash</code>
Apply rule to folders in hosted domain <code>siroe.com</code>	<code>user/*@siroe.com/*</code>	<code>user/.*@siroe.com/.*</code>
Apply rule to folders in default domain.	Not applicable	<code>user/^.*/.*</code>

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 `schedule.task:expire.crontab` option as described in [Expire and Purge Log and Scheduling Options](#).

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 [Scheduling Automatic Tasks](#)). For example:

```
./msconfig
msconfig> set schedule.task:expire.crontab "0 1 * * 6 bin/imexpire -e"
msconfig# write
msconfig> exit
```

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 Log and Scheduling Parameters

For Purge options, see the topic on maintenance queue `configutil` parameters in *Messaging Server System Administrator's Guide*. Clicking on the specific parameter/option will often provide both the legacy (`configutil`) parameter and the unified configuration (`msconfig`) option.

Expire and Purge Log and Scheduling Options

Parameter	Description
<code>schedule.task:expire.enable</code>	Whether the expire task should be scheduled. Default: 1
<code>schedule.task:expire.crontab</code>	<p>Interval for running <code>imexpire</code>. Uses UNIX <code>crontab</code> format: <i>minute hour day-of-month month-of-year day-of-week</i></p> <p>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.</p> <p>You can also use the <code>-e</code> and <code>-c</code> flags with <code>imexpire</code> to and expire only or purge only respectively.</p> <p>Interval Examples:</p> <ol style="list-style-type: none"> 1. Run <code>imexpire</code> at 12:30am, 8:30am, and 4:30pm: <code>30 0,8,16 * * * bin/imexpire</code> 2. Run <code>imexpire</code> at weekday morning at 3:15 am: <code>15 3 * * 1-5 bin/imexpire</code> 3. Run <code>imexpire</code> only on Mondays: <code>0 0 * * 1 bin/imexpire</code> Default: <code>0 23 * * * bin/imexpire</code> <p>To disable: Run <code>msconfig set schedule.task:expire.enable 0</code></p>
<code>store.expire.exploglevel</code>	<p>Specify a log level:</p> <p>0 = No log. 1 = Log summary for the entire expire session. 2 = Log one message per mailbox expired. 3 = Log one message per message expired. Default: 0</p>

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 configuration option `store.expire.exploglevel` can be set to 0, 1, 2, or 3 for different levels of logging. Loglevel 0 is the default, and no logging is performed. Loglevel 1 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.

Configuring POP, IMAP, and HTTP Services in Unified Configuration

Configuring POP, IMAP, and HTTP Services in Unified Configuration

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](#)
- [Login Requirements](#)
- [Performance Options](#)
- [Client Access Controls](#)
- [To Configure POP Services](#)
- [To Configure IMAP Services](#)
- [To Configure HTTP Services](#)

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](#)
- [Specifying Port Numbers](#)
- [Ports for Encrypted Communications](#)
- [Service Banner](#)

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 in Unified Configuration](#).)

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 in Unified Configuration](#).

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 `msconfig` utility and the `(pop.banner)` option to set service banners.

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 do 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:

```
./msconfig set base.loginseparator "+"
```

2. Inform POP client users that they should log in 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 the `auth.searchfordomain` option 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 `auth.searchfilter` option. This option is a global default setting for the `inetDomainSearchFilter` per-domain attribute and follows the same syntax.

Refer to http://msg.wikidoc.info/index.php/Configutil_Reference for further information on these options.

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 Options

You can set some of the basic performance options for the POP, IMAP, and HTTP services of Messaging Server. Based on your hardware capacity and your user base, you can adjust these options 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.

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.

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.

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 in Unified Configuration](#).

To Configure POP Services

You configure the Messaging Server POP service by using the `msconfig` 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 the POP service:

```
msconfig set pop.enable 1
```

- To disable the POP service:

```
msconfig set pop.enable 0
```

- To specify the port number:

```
msconfig set pop.port <port number>
```

- To set the maximum number of network connections per process (see [Number of Connections per Process](#) for details):

```
msconfig set pop.maxsessions <number>
```

- To set the maximum idle time for connections (see [Dropping Idle Connections](#) for details):

```
msconfig set pop.idletimeout <number>
```

- To set the maximum number of threads per process (see [Number of Threads per Process](#) for more information):

```
msconfig set pop.maxthreads <number>
```

- To set the maximum number of processes (see [Number of Processes](#) for additional information):

```
msconfig set pop.numprocesses <number>
```

- To enable POP over SSL on port 995:

```
# ./msconfig
msconfig> set pop.enablesslport 1
msconfig# set pop.sslusessl 1
msconfig# set pop.sslport 995
msconfig# write
msconfig> exit
# ./stop-msg pop
# ./start-msg pop
```

TLS is also supported if SSL is configured correctly.

- To specify a protocol welcome banner:

```
msconfig set pop.banner <banner>
```

To Configure IMAP Services

You configure the Messaging Server IMAP service by using the `msconfig` 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](#)

- To enable the IMAP service:

```
msconfig set imap.enable 1
```

- To disable the IMAP service:

```
msconfig set imap.enable 0
```

- To specify the port number:

```
msconfig set imap.port <number>
```

- To enable a separate port for IMAP over SSL:

```
msconfig set imap.enablesslport 1
```

- To specify a port number for IMAP over SSL:

```
msconfig set imap.sslport <number>
```

- To enable or disable password login to the IMAP service:

```
msconfig set imap.plaintextmincipher <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):

```
msconfig set imap.maxsessions <number>
```

- To set the maximum idle time for connections (see [Dropping Idle Connections](#) for additional information):

```
msconfig set imap.idletimeout <number>
```

- To set the maximum number of threads per process (see [Number of Threads per Process](#)):

```
msconfig set imap.maxthreads <number>
```

- To set the maximum number of processes (see [Number of Processes](#)):

```
msconfig set imap.numprocesses <number>
```

- To specify a protocol welcome banner:

```
msconfig set imap.banner <banner>
```

- To enable IMAP over SSL on port 993:

```
# ./msconfig
msconfig> set imap.enablesslport 1
msconfig# set imap.sslusessl 1
msconfig# set imap.sslport 993
msconfig# write
msconfig> exit
# ./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 in Unified Configuration](#) 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.

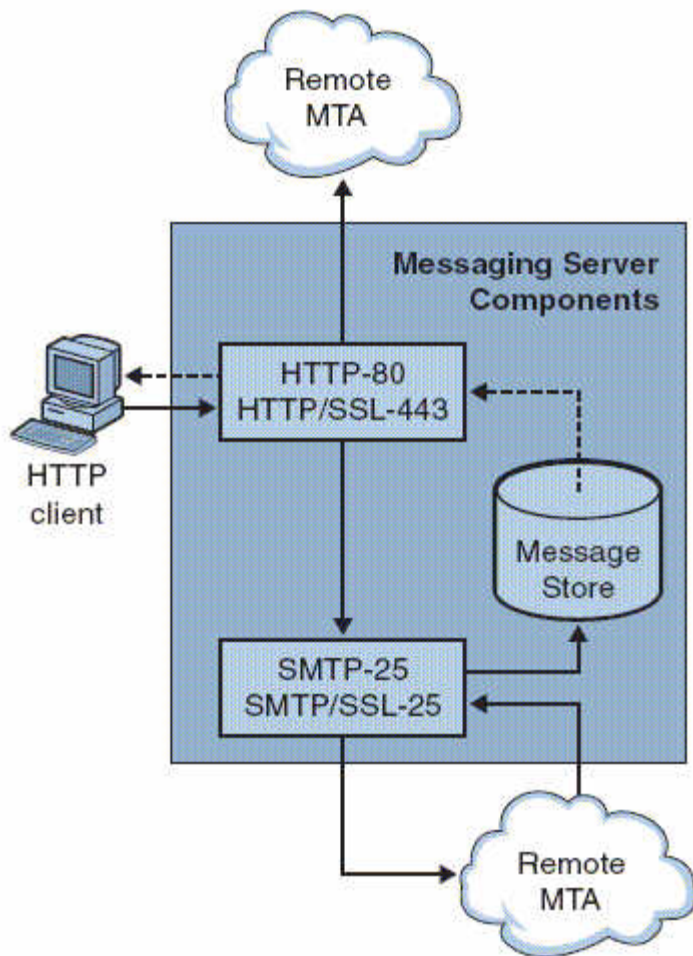


Note

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 `mshhttpd` 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 options are similar to the options available for the POP and IMAP services, including options 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 for user log-in. The configuration options to do this are as follows:

- `base.proxyimapport`: IMAP port on which to connect (default 143).
- `base.proxyimapssl`: Enable SSL (default no).
- `base.proxyadmin`: Specifies the store Admin ID.
- `base.proxyadminpass`: Specifies the store Admin password.

You can set these options globally in Unified Configuration to apply to every IMAP back-end server by using `base.proxyadmin`. Alternatively, you can set these options for each individual IMAP back-end server by using `proxy:storeaffinitygroup.imapadmin`.

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 `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 `base.proxyadmin` and `base.proxyadminpass` are 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 the HTTP service:

```
msconfig set http.enable 1
```

- To disable the HTTP service:

```
msconfig set http.enable 0
```

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:

```
msconfig set http.port <number>
```

- To enable a separate port for HTTP over SSL:


```
msconfig set http.enablesslport 1
```

- To specify a port number for HTTP over SSL:

```
msconfig set http.sslport <number>
```

- To enable or disable password login:

```
msconfig set http.plaintextmincipher <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](#)):

```
msconfig set http.maxsessions <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](#)):

```
msconfig set http.sessiontimeout <number>
```

- To set the maximum number of threads per process:

```
msconfig set http.maxthreads <number>
```

- To set the maximum number of processes:

```
msconfig set http.numprocesses <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 pool directory or specify an alternate directory. You can also specify a maximum size allowed for attachments. To specify the attachment pool 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 percent more space. Thus, a 5 Mbyte limit in the option results in the maximum size of one message and attachments being about 3.75 Mbyte.

- To set the pool directory:

```
msconfig set http.spooldir <dirpath>
```

- To specify the maximum message size:

```
msconfig set http.maxmessagesize <size>
```

where *size* is a number in bytes. 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 option results in the maximum size of one message and attachments being about 3.75 Mbytes.

- To specify an alternate MTA host name:

```
msconfig set http.smtphost <hostname>
```

- To specify the port number for the alternate MTA host name:

```
msconfig set http.smtpport <portnum>
```

To enable HTTP access over SSL on port 8991:

```
# ./msconfig
msconfig> set http.enablesslport -1
msconfig# set http.sslusessl 1
msconfig# set http.sslport 8991
msconfig# write
msconfig> exit
# ./stop-msg http
# ./start-msg http
```

Handling Message Store Overload in Unified Configuration

Handling Message Store Overload in Unified Configuration

The following features described here were introduced in **Messaging Server 7.0**.

Topics:

- [Overview of Managing Message Store Load](#)
- [Message Store Load Throttling](#)
- [Job Controller Stress Handling](#)
- [Default Job Controller Configuration](#)

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.

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 `job_controller.stressblackout` seconds of a previous stressed message for the same channel. If the message is processed, then the job controller multiplies the effective `threaddepth` option for the channel by `job_controller.stressfactor`, and subtracts `job_controller.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 the Job Controller asks all current master programs for the channel to exit, and, if the queue is not empty, starts an appropriate number of processes.

When `job_controller.stresstime` seconds has passed after the last stress change, the Job Controller divides `threaddepth` by `job_controller.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.

Default Job Controller Configuration

These configuration options have the following default values:

```
job_controller.stressblackout=60
job_controller.stresstime=120
job_controller.stressfactor=5
job_controller.stressjobs=2
job_controller.unstressfactor=stressfactor
job_controller.unstressjobs=stressjobs
```

Managing Message Store Partitions and Adding Storage in Unified Configuration

Managing Message Store Partitions and Adding Storage in Unified Configuration

This information describes message store partitions and adding storage.

Topics:

- [Message Store Partition Overview](#)
- [To Add a Message Store Partition](#)
- [To Change the Default Message Store Partition](#)
- [Adding More Physical Disks to the Message Store](#)

See also the topic on moving mailboxes to a different disk partition in *Messaging Server System Administrator's Guide*.

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 the topic on message store directory layout in *Messaging Server System Administrator's Guide*). 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 enables 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 Messaging Server to refresh the configuration information.

- **Command Line**, To add a partition to the store at the command line:
The location of message files is controlled by setting the `partition:partition_name` `.messagepath` option, where `partition_name` is the logical name of the partition.

```
msconfig set partition:<partition_name>.messagepath <path>
```

`path` indicates the absolute path name where the partition is stored.
To specify the path to the primary partition:

```
# msconfig set partition:primary.path <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 `store.defaultpartition` option.

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 to add more stores as needed. See [Using Sun StorageTek 53xx NAS with Messaging Server Message Store](#) or the topic on using NetApp filers with Messaging Server message store in *Messaging Server System Administrator's Guide* for information on how to add disks to the message store.

Managing Message Store Quotas in Unified Configuration

Managing Message Store Quotas in Unified Configuration

This information describes the quota tasks. See [Message Store Quota \(Overview\) in Unified Configuration](#) for conceptual information.

Topics:

- [To Specify a Default User Quota](#)
- [To Specify Individual User Quotas](#)
- [To Specify Domain Quotas](#)
- [To Set Up Quota Notification](#)
- [To Enable or Disable Quota Enforcement](#)
- [To Set a Grace Period](#)
- [Netscape Messaging Server Quota Compatibility Mode](#)

See also "Monitoring the Message Store" in *Messaging Server System Administrator's Guide*.

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
2. Specifying which users are bound to the default quota

The following examples show how to set default user quotas. Refer to [Messaging Server Reference](#) for detailed option information.

- To specify a default user disk quota for message size in bytes:

```
msconfig set store.defaultmailboxquota [ -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:

```
msconfig set store.defaultmessagequota [ -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.

In addition, a number of configuration options of the form `store.quota*` can be used to implement more finely grained quota policies. See [Messaging Server 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`.

The following LDAP entry specifies 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` documentation in *Unified Communications Suite Schema Reference* for more information about **mailQuota values**. Also see [Managing Message Types in the Message Store in Unified Configuration](#).

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 in the desired LDAP domain entry.

- 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 Set Up Quota Notification

Quota notification is the process of sending users a warning message when they are getting close to their quota.

1. To enable quota notification, run the following command:

```
msconfig set store.quotanotification 1
```

If the message is not set, no quota warning message is sent to the user.

2. To define a quota warning message, run the following command:

```
msconfig set store.quotaexceededmsg '<message>'
```

For example:

```
msconfig set store.quotaexceededmsg 'Subject: WARNING: User quota
exceeded$$User quota threshold exceeded - reduce space used.'
```

The Warning Message is the message that is sent to users who are close to exceeding their disk quota. The warning message must:

- Be in RFC 822 format
- Contain a header with at least a subject line, followed by \$\$, then the message body
- Use "\$" to represent 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.)

You can also use the following variables in the message:

- [ID] - userid
- [DISKUSAGE] - disk usage
- [NUMMSG] - number of messages
- [PERCENT] - store.quotawarn percentage
- [QUOTA] - mailquota attribute
- [MSGQUOTA] - mailmsgquota attribute

The following example shows a warning message that uses these variables:

```
msconfig set store.quotaexceededmsg 'Subject: Overquota
Warning$$[ID],$$Your mailbox size has exceeded [PERCENT] of its
allotted quota.$Disk Usage: [DISKUSAGE]$Number of Messages:
[NUMMSG]$Mailquota: [QUOTA]$Message Quota:
[MSGQUOTA]$$-Postmaster'
```

3. To specify how often the warning message is sent, run the following command:

```
msconfig set store.quotaexceededmsginterval <number>
```

where *number* indicates a number of days. For example, 3 would mean the message is sent every 3 days.

4. To specify a quota threshold, run the following command:

```
msconfig set store.quotawarn <number>
```

where *number* indicates a percentage of the allowed quota.

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 the `store.quotaoverdraft` is enabled, email notifications are not triggered until the user's disk usage exceeds 100 percent 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 Disable Quota Notification

- To disable quota notification, run the following command:

```
msconfig set store.quotanotification 0
```

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

- To enable quota enforcement at the user level, run the following command:

```
msconfig set store.quotaenforcement 1
```

The MTA saves over-quota messages in its queues and notifies users that their messages were 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:

```
msconfig set store.overquotastatus 1
```

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 options are disabled or not set:

- `store.quotaenforcement`
- `store.overquotastatus`
- `store.quotaoverdraft`

When `store.overquotastatus` is enabled (set to 1), it always treats `store.quotaoverdraft` as enabled, otherwise users never go over quota to trigger the rejection. Also, when `store.quotaoverdraft` is enabled, 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 options, restart Messaging Server.

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:

```
msconfig set store.quotagraceperiod <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 an option, `store.quotaoverdraft`, which retains this behavior.

When `store.quotaoverdraft` is enabled (set to 1), 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 option is disabled (set to 0).

Managing Message Types in the Message Store in Unified Configuration

Managing Message Types in the Message Store in Unified Configuration

This information describes how to work with message types. For conceptual information, see [Message Store Message Types Overview in Unified Configuration](#).

Topics:

- [To Configure Message Types](#)
- [Sending Notification Messages for Message Types](#)
- [Administering Quotas by Message Type](#)
- [Expiring Messages by Message Type](#)

To Configure Message Types

To configure a message type, use the `msconfig` utility to set the `store.messageType` option values that define and identify the message type.

1. Enable message types by setting the `store.messageType.enable` option to 1. This option enables the message store to identify and manipulate message types. You must set this option before you can configure an individual message type. For example, type the following command:

```
msconfig set store.messageType.enable 1
```

2. Define and identify the message type by setting `store.messageType.mtindexx` options. 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 option 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:

```
msconfig set store.messageType.mtindex:1.contentType text/plain
```

- To define a voice message type, type the following command:

```
msconfig set store.messageType.mtindex:2.contentType  
multipart/voice-message
```

3. Provide a flag name for the message type by setting the `store.messageType.mtindex:x.flagname` option. This option 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.mtindex:`

`x` option. The variable `x` is the integer ID of the message type defined with the `store.messageType.mtindex:x` option. For example, to define flag names for the message types configured in the preceding step, type the following commands:

```
msconfig set store.messageType.mtindex:1.flagname text
msconfig set store.messageType.mtindex:2.flagname voice_message
```

4. Configure a quota root name for the message type by setting the `store.messageType.mtindex:x.quotaroot` option. 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:

```
msconfig set store.messageType.mtindex:1.quotaroot text
msconfig set store.messageType.mtindex:2.quotaroot 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 `X-Message-Type`, enter the following command:

```
msconfig set store.messageType.header X-Message-Type
```

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 in Unified Configuration \(Tasks and Examples\)](#).

To enable the JMQ notification plug-in to recognize a particular message type, you must configure the `store.messageType` options, including the `store.messageType.mtindex:x.flagname` option. 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 users informing them that their inbox has new voice mail and text messages.

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 options:

- Set the `store.messageType.mtindex:X.quotaroot` option for each message type. For details, see [To Configure Message Types](#).
- Set the `store.typequota.enable` option to 1.
For example, type the following command:

```
msconfig set store.typequota.enable 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 in Unified Configuration](#).

When you set a quota for the message type with a `msconfig` option or LDAP attribute, you must use the quota root specified with the `store.messageType.mtindex:X.quotaroot` option.

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 MBytes.
- The default mailbox message quota is 5000.
- The storage quota for the Archive folder is 100 MBytes.
- The storage quota for text message types is 10 MBytes.
- The message quota for text message types is 2000.
- The storage quota for voice message types is 10 MBytes.
- The message quota for voice message types is 200.

This quota root permits greater storage in the Archive folder (100 MBytes) than in all the other folders and message types combined (60 MBytes). 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:

1. Configure the `store.messageType.mtindexX.quotaRoot` option as follows:

```
msconfig set store.messageType.mtindex:1.quotaRoot text
msconfig set store.messageType.mtindex: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 enables 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 in Unified Configuration \(Tasks\)](#).

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 following example assumes you have configured text and voice message types as follows:


```
store.messageheader.mtindex:1 text/plain
store.messageheader.mtindex: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 in Unified Configuration

Managing Shared Folders in Unified Configuration

This information describes the tasks that you use to administer shared folders. See the topic on message store shared folders overview in *Messaging Server System Administrator's Guide* for conceptual information.

Topics:

- [Specifying Sharing Attributes for Private Shared Folders](#)
- [To Create a Public Shared Folder](#)
- [To Grant Folder Access Rights Based on Group Membership](#)
- [To Set or Change a Shared Folder's Access Control Rights](#)
- [Enabling or Disabling Listing of Shared Folders](#)
- [Setting Up Distributed Shared Folders](#)
- [Monitoring and Maintaining Shared Folder Data](#)

Specifying 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
- Convergence web client
- Messaging Server `readership` command, for mail administrators

The following table explains the `msconfig` options that pertain to private shared folders:

<code>msconfig</code> Option	Description	Default
<code>store.privatesharedfolders.restrictanyone</code>	If enabled (1), disallows regular users from setting the permission on private shared folders to <code>anyone</code> .	0
<code>store.privatesharedfolders.restrictdomain</code>	If enabled (1), disallows regular users sharing private folders to users outside of their domain.	0
<code>store.privatesharedfolders.shareflags</code>	If disabled (0), users of a shared folder have their own set of flags (for example, <i>seen</i> , <i>deleted</i> , 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` option specifies the `userid` to act as a container for all public shared folders (see the topic on message store shared folders overview in *Messaging Server System Administrator's Guide* for more information.). Typically, this is simply `public`. The default is `NULL` (unset).

```
msconfig set store.publicsharedfolders.user 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 by using the `mboxutil` command, for example:

```
mboxutil -c user/public/gardening
```

4. Use the `readership` command to 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.

The following command grants 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.

See the `readership` command in *Messaging Server System Administrator's Guide* for a list of 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 by using a mail client, or the mail administrator could have created it by using the `mboxutil` command, for example:

```
mboxutil -c user/gregk/gardening
```

If the team 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 by using Convergence. 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](#) later in this information.



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.

Shared Folder Examples

- To assign everyone in the `sesta` domain to have lookup, read, and email marking (but not posting) access to the public folder called `golftournament`, type the following command:

```
readership -s user/public/golftournament anyone@sesta lwr
```

- To assign the same access to everyone on the message store type the following command:

```
readership -s user/public/golftournament anyone lwr
```

- To assign lookup, read, email marking, and posting rights to a group, type the following command:

```
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`, type the following command:

```
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`, type the following command:

```
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`, type the following command:

```
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 by 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](#).

Enabling or Disabling Listing of Shared Folders

Use the `store.sharedfolders` option to enable or disable listing of shared folders when responding to an IMAP `LIST` command. Setting the option to `0` disables it. The setting is enabled by default (set to `1`). `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.

Setting Up Distributed Shared Folders

Normally, shared folders are only available to users on a particular message store. Messaging Server, however, enables 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. However, web mail clients do not support remote shared folders access. Users can list and subscribe to the folders, but they cannot view or alter the contents.

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.

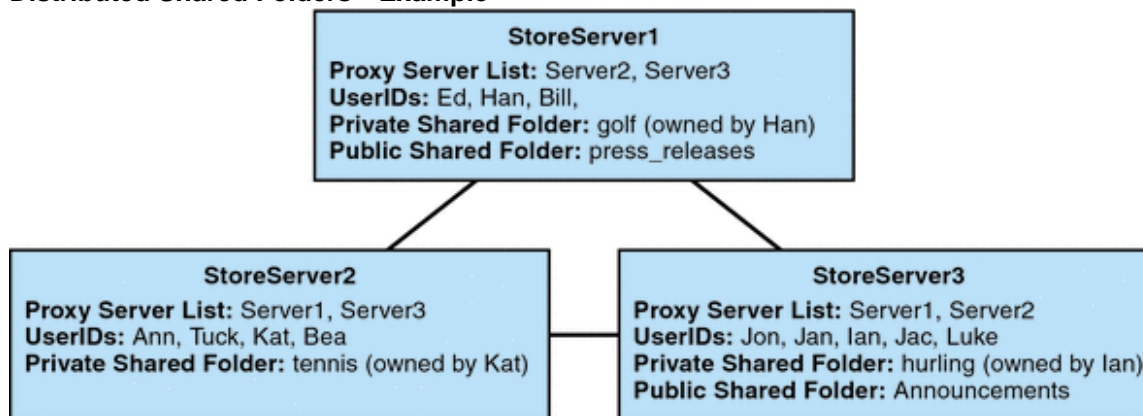
Variables for Configuring Distributed Shared Folders

Name	Value	Data Format
<code>base.proxyserverlist</code>	Message store server list to list shared folders from	space-separated strings
<code>base.proxyadmin</code>	Default store admin login name	string
<code>base.proxyadminpass</code>	Default store admin password	string
<code>proxy:hostname</code> <code>.imapadmin</code>	Store admin login name for a specific host if different from <code>base.proxyadmin</code>	string
<code>proxy:hostname</code> <code>.imapadminpass</code>	Store admin password for a specific host if different from <code>base.proxyadminpass</code>	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 `msconfig` options. 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
```

Monitoring and Maintaining Shared Folder Data

The `readership` command-line utility enables 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
<code>-d days</code>	Returns a report, per shared folder, of the number of users who have selected the folder within the specified days.
<code>-p months</code>	Removes data from the <code>peruser.db</code> for those users who have not selected their shared folders within the specified months.
<code>-l</code>	List the data in <code>lright.db</code> .
<code>-s</code> <code>folder_identifier_rights</code>	Sets access rights for the specified folder. This updates the <code>lright.db</code> as well as the <code>folder.db</code> .

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, use the following 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:

```
readership -d 30
```


To List Users and Their Shared Folders

To list users and the shared folders to which they have access, use the following 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 and other folders in a specified time period), use the following commands:

1. This command writes the inactive mailboxes to a file:

```
mboxutil -o [-w <file>] [-t <number of days>]
```

2. This command removes the mailboxes in a given file:

```
mboxutil -d -f <file>
```

Example: Remove users who have not accessed folders for the past six months (180 days) using a file named `inactive_users`:

```
mboxutil -o -w inactive_users -t 180
mboxutil -d -f inactive_users
```

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 in Unified Configuration

Message Store Administration in Unified Configuration

Below is a dynamically generated list of administering message store in unified configuration pages on this wiki. See also the main [message store in unified configuration](#) page.

Page: [Protecting Mailboxes from Deletion or Renaming \(Unified Configuration\)](#)

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: [Managing Message Store Quotas in Unified Configuration](#)

Page: [Message Store and Mailbox Management in Unified Configuration](#)

Message Store Architecture and Concepts in Unified Configuration

Message Store Architecture and Concepts in Unified Configuration

- [Valid Message Store UIDs and Folder Names in Unified Configuration](#)
- [Message Removal in Unified Configuration](#)
- [Message Store Message Types Overview in Unified Configuration](#)
- [Automatic Recovery on Startup in Unified Configuration](#)
- [Very Large Mailboxes](#)
- [Automatic Message Deletion](#)
- [Quota](#)
- [Command-line Utilities](#)
- [Logging](#)
- [Load Throttling](#)
- [Best Practices for Oracle Communications Messaging Server and Oracle Solaris ZFS in Unified Configuration](#)

See also the following topics in *Messaging Server System Administrator's Guide*:

- [Upgrading the Message Store](#)
- [Message Store Directory Layout](#)
- [Shared Folders](#)

Message Store Automatic Recovery On Startup in Unified Configuration

Message Store Automatic Recovery On Startup in Unified Configuration

This information provides a conceptual overview of the startup and automatic recovery process of `stored`, and message store database snapshot. For more information on related tasks, see the topic on administering message store database snapshots in *Messaging Server System Administrator's Guide*.

Topics:

- [Overview of Automatic Recovery on Startup](#)
- [Automatic Startup and Recovery Theory of Operations](#)

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 are indicated in the default log file, and almost always are 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 in Unified Configuration](#).

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 by writing 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 indicate that `reconstruct` should be run 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 are useful to administrators and some messages consist of coded data to be used for internal analysis. If `stored` detects any problems, it attempts to fix the database and try starting it again.

When the database is opened, `stored` signals that the rest of the services may start. If the automatic fixes failed, messages in the default log specify what actions to take. See [Error Messages Signifying that reconstruct Is Needed](#).

After most recoveries, the database is usually be up-to-date and no further action is required. However,

some recoveries require a `reconstruct -m` to synchronize redundant data in the message store. Again, this is stated in the default log, so it is important to monitor the default log after a startup. Even though the message store seems 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 should 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. For more information on related tasks, see the topic on administering message store database snapshots in *Messaging Server System Administrator's Guide*.

A snapshot is a hot backup of the database and is used by `stored` to restore a broken database transparently. This is much quicker than using `reconstruct` to rebuild the entire database from scratch with the information stored in the message and index partitions.

Snapshots of the database are taken automatically by the scheduler. The default snapshot schedule consists of a full snapshot every day and incremental snapshots every 10 minutes. (Note that older versions of the messaging server have a more frequent default incremental snapshot schedule).

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.

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 is automatically be recovered. Two snapshot options can be configured: the location of the snapshot file and number of snapshots saved. These options are shown in [Message Store Database Snapshot Parameters](#).

Having a snapshot interval which is too small results in a frequent burden to the system and a greater chance that a problem in the database is copied as a snapshot. Having a snapshot interval too large can create a situation where the database holds the state it had back when the snapshot was taken.

`stored` monitors the database and is intelligent enough to refuse the latest snapshot if it suspects the database is not perfect. It instead retrieves the latest most reliable snapshot. Despite the fact that a snapshot may be retrieved from a day ago, the system uses more up to date redundant data and overrides 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 Options

The following table shows the snapshot options that you set with the `msconfig` command.

Message Store Database Snapshot Options

Parameter	Description
<code>store.snapshotpath</code>	Location of message store database snapshot files. Either existing absolute path or path relative to the <code>store</code> directory. Default: <code>dbdata/snapshots</code>
<code>store.snapshotdirs</code>	Number of different snapshots kept. Valid values: 2 -367 Default: 3

Message Store Maintenance Queue in Unified Configuration

Message Store Maintenance Queue in Unified Configuration

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.

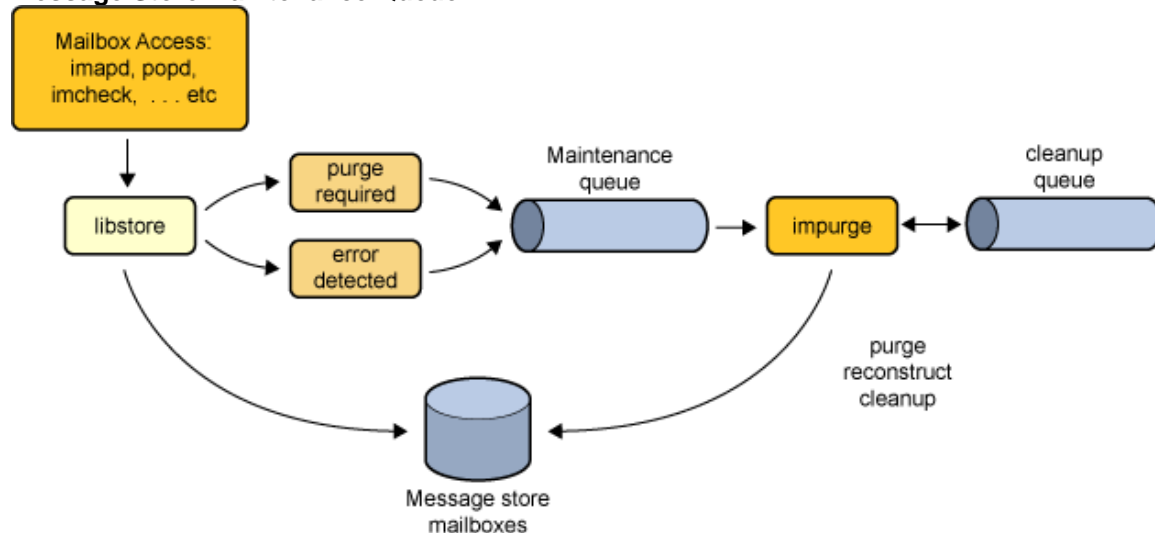
Topics:

- [Message Store Maintenance Queue Overview](#)
- [Displaying the Maintenance Queue](#)
- [Deleting, Expunging, Purging, and Cleaning Up Messages](#)
- [Mailbox Self Healing \(Auto Repair\)](#)
- [Maintenance Queue Configuration Options](#)
- [The impurge Command](#)

Message Store Maintenance Queue Overview

The following diagram summarizes the major components and their interactions with the queue.

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` option). 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 `mbxlist` database. Mailboxes that have already

been repaired after the enqueue time are ignored. Obsolete 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 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 `imap.immediateflagupdate` option 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 mailbox` 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%). This is different from the `imsimta purge` command, 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, an 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/2012:09:55:57 +1100] hostname imapd[7264]: Store Critical:
Unable to open mailbox user/username: Mailbox does not exist
[25/Feb/2012:09:55:57 +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 has been processed and the `store.idx` restored:

```
[25/Feb/2012:09:55:58 +1100] hostname impurge[7263]: General Notice:
repairing user/username
```

Maintenance Queue Configuration Options

The `store.purge.enable` option 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 [Messaging Server 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 `store.purge.enable` option 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/2012:14:47:15 +1100] hostname impurge[17986]: General Error:
Could not get purge session lock.
Possibly another impurge is running
```

For details, see the `impurge` documentation in *Messaging Server System Administrator's Guide*.

Message Store Message Types Overview in Unified Configuration

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 in Unified Configuration](#) for task information.

Topics:

- [About Message Type](#)
- [Planning the Message-Type Configuration](#)
- [Defining and Using Message Types](#)

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 `msconfig` command to set values for the `store.messagestype` option. 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.messagestype.mtindex:n.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](#).
- 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](#).

The message-type user flag is read only. It cannot be modified by IMAP commands.

The following examples assume that you configure the message-type `msconfig` options with the values

shown here:

```
store.message_type.enable 1
store.message_type.mtindex:1.contenttype text/plain
store.message_type.mtindex:1.flagname text
store.message_type.mtindex:1.quotaroot text
store.message_type.mtindex:2.contenttype multipart/voice-message
store.message_type.mtindex:2.flagname voice_message
store.message_type.mtindex:2.quotaroot voice
```

Example 1: IMAP FETCH Session Based on the Message-Type `msconfig` 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.message_type.mtindex:n.flagname` option.

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 `msconfig` 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.message_type.mtindex:2.flagname` option.

Message Store Quota (Overview) in Unified Configuration

Message Store Quota (Overview) in Unified Configuration

This information describes quota concepts. See [Managing Message Store Quotas in Unified Configuration](#) 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](#)
- [Quota Theory of Operations](#)
- [Message Store Quota Attributes and Options](#)

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 on 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 number of 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 the MTA immediately, if `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 the topic on message store message expiration in *Messaging Server System Administrator's Guide*).

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 `msconfig` options 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 the quota attributes and `msconfig` options will take effect automatically, but not immediately as information is stored in caches and it may take a little time before the changes fully take effect. 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 Options

This section lists the major the message store quota attributes and `msconfig` options. The intention is to provide you with an overview of the functionality interface. For detailed information on these attributes and options, refer to the appropriate reference documentation.

The following table lists the quota attributes. Refer to *Unified Communications Suite Schema Reference* for more information.

Message Store Quota Attributes

Attribute	Description
<code>mailQuota</code>	Bytes of disk space allowed for the user's mailbox.
<code>mailMsgQuota</code>	Maximum number of messages permitted for a user. This is a cumulative count for all folders in the store.
<code>mailUserStatus</code>	Status of the mail user. Some of the possible values are <code>active</code> , <code>inactive</code> , <code>deleted</code> , <code>hold</code> , and <code>overquota</code> .
<code>mailDomainDiskQuota</code>	Bytes of disk space allowed for the cumulative count of all the mailboxes in a domain.
<code>mailDomainMsgQuota</code>	Maximum number of messages permitted for a domain, that is, the total count for all mailboxes in the store.
<code>mailDomainStatus</code>	Status of the mail domain. Values and default are the same as <code>mailUserStatus</code> .

To have the preceding attributes take effect, run `iminitquota` to make quota and usage up-to-date. The changes would take effect without running this, but not immediately, as information is stored in caches and it takes a little time before the changes take effect.

The following table lists the quota options. Refer to [Overview of Messaging Server Unified Configuration](#) for the latest and most detailed information.

Message Store `msconfig` Options

Option	Description
<code>store.quotaenforcement</code>	Enable quota enforcement. When off, the quota database is still updated, but messages are always delivered. Default: 1 (bool).
<code>store.quotanotification</code>	Enable quota notification. Default: 0 (bool).
<code>store.defaultmailboxquota</code>	Store default quota by number of bytes. Default: "-1" (string) (unlimited).
<code>store.defaultmessagequota</code>	Store default quota by number of messages. Numeric. Default: "-1" (string) (unlimited).
<code>store.quotaexceededmsg</code>	Message to be sent to user when quota exceeds <code>store.quotawarn</code> . If none, notification is not sent. Default: No default (non-empty string). 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] - <code>store.quotawarn</code> percentage, [QUOTA] - mailbox quota attribute, [MSGQUOTA] - mailboxquota attribute.
<code>store.quotaexceededmsginterval</code>	Interval, in days, for sending overquota notification. Default: 7 (int32).
<code>store.quotagraceperiod</code>	Time, in hours, a mailbox has been overquota before messages to the mailbox will bounce back to the sender. Number of hours. Default: 120 (uint32).
<code>store.quotawarn</code>	Quota warning threshold. Percentage of quota exceeded before clients are sent an over quota warning. Default: 90 (int32).
<code>store.quotaoverdraft</code>	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: 0 (bool), but is treated as on if <code>store.overquotastatus</code> is set, otherwise the user can never go over quota and the <code>overquotastatus</code> is never used.
<code>store.overquotastatus</code>	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 <code>mailuserstatus</code> LDAP attribute is set to overquota so no more messages are accepted by the MTA. Default: 0 (bool).

To have the preceding `msconfig` options 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.

Migrating Mailboxes to a New System in Unified Configuration

Migrating Mailboxes to a New System in Unified Configuration

Messaging Server provides several ways to move or relocate mailboxes from one Messaging Server host (mailhost) to another.

Topics:

- [Tools Summary for Relocating Messaging Server Users to a New Mailhost](#)
- [Migrating Mailboxes from an x86 Host to a SPARC Host](#)
- [Moving Mailboxes to Another Messaging Server While Online](#)
- [To Move Mailboxes Using an IMAP client](#)
- [To Move Mailboxes by Using the `MoveUser` Command](#)
- [To Move Mailboxes by Using the `imsimport` Command](#)

For more information, see the documentation for the `moveruser` and `rehostuser` commands in *Messaging Server System Administrator's Guide*.

Tools Summary for Relocating Messaging Server Users to a New Mailhost

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.
- **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.
- **imsbackup | imsrestore**: Manually backs up the account from one host and restores on another. Can be combined 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.

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 (as show in the Message Store Components matrix in *Messaging Server System Administrator's Guide*) 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 prior 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 Online

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 online 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 the topic on monitoring user access to the message store in *Messaging Server System Administrator's Guide*.)
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.

```
msconfig set mta.alias_entry_cache_timeout 0
```

- a. On the back-end message stores containing the mailboxes to be moved, set the authentication cache timeout to 0.

```
msconfig set base.authcachettl -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.
For the IMAP proxy and POP proxy, set both `ldapcachettl` and `authcachettl` to 0.
For example:

```
./msconfig
msconfig> set imapproxy.authcachettl 0
msconfig# set imapproxy.ldapcachettl 0
msconfig# set popproxy.authcachettl 0
msconfig# set popproxy.ldapcachettl 0
msconfig# write
```

- 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 Server hosts that run an MTA that inserts messages into the migrating mailboxes are typically the back-end message store. However, if the system is using LMTP, then that system is the inbound MTA. Check your configuration to make sure.
Resetting the `alias_entry_cache_timeout` option 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 than the out-of-date cached information.
- 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 are 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. 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 in Unified Configuration](#).

**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 `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 to 900 as follows:

```
msconfig set base.authcachettl 900
```

For the IMAP proxy and POP proxy, set both `ldapcachettl` and `authcachettl` to 900. For example:

```
./msconfig
msconfig> set imaproxy.authcachettl 900
msconfig# set imaproxy.ldapcachettl 900
msconfig# set popproxy.authcachettl 900
msconfig# set popproxy.ldapcachettl 900
msconfig# write
```

For MTAs, set the `alias_entry_cache_timeout` option to 600.

```
msconfig set mta.alias_entry_cache_timeout 600
```

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 that 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 options. 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.
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` documentation in *Messaging Server System Administrator's Guide*.)

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 `store.relinker.enable` to 1.

This reduces the message store size on the new system caused by duplicate storage of identical messages.

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 is 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 `store.relinker.enable` to 1.
This reduces the message store size on the new system caused by duplicate storage of identical messages.
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 the `moveuse` documentation in *Messaging Server System Administrator's Guide* 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 Messaging Server message store. However, if the Messaging Server host 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 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 `store.relinker.enabled` to 1.
This reduces the message store size on the new system caused by duplicate storage of identical messages.
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` documentation in *Messaging Server System Administrator's Guide* 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 in Unified Configuration

Monitoring Disk Space in Unified Configuration

This section describes configuration options for monitoring disk and partition usage and for generating warnings about disk space availability.

Topics:

- [Overview](#)
- [Symptoms of Insufficient Disk Space](#)
- [Monitoring Disk Space](#)
- [Monitoring the Message Store](#)

Overview

Inadequate disk space or inadequate space within a disk partition are among the most common causes of mail server problems and failure. Typical causes of inadequate space are:

- Message store quotas are not enforced and the message store outgrows the disk space available for a partition.
- Over-long MTA message queues.
- Log files that are not adequately monitored and kept within defined limits. (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 Insufficient Disk Space

Symptoms of insufficient disk-space are:

- MTA queues overflow and reject SMTP connections.
- Messages remain in the `ims-master` queue and are not delivered to the message store.
- Log files 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 cause down time.

Monitoring 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 options to monitor message store disk usage
- Lock message store partitions when a disk-usage threshold is reached

Monitoring the Message Store

You can monitor message store disk usage by configuring the following attributes with the `msconfig` utility:

- `alarm.system:<alarmtype>.statinterval`
Specifies the length of time, in seconds, between disk availability checks. For example, to set the system to monitor disk space every 600 seconds, enter the following command:

```
msconfig set alarm.system:diskavail.statinterval 600
```

- `alarm.system:<alarmtype>.threshold`
Specifies a percentage of disk space that must be available or a warning is generated. For example, it is recommended that disk-space usage should not exceed 75%; correspondingly, the following command generates a warning whenever the amount of disk space available falls below 25%:

```
msconfig set alarm.system:diskavail.threshold 25
```

- `alarm.system:<alarmtype>.warninginterval`
Specifies an interval, in hours, between the repetition of disk availability alarms. For example, the following command sets an interval of one hour between one disk availability warning and another.

```
msconfig set alarm.system:diskavail.warninginterval 1
```

- `alarm.system:<alarmtype>.description`
A description of the disk availability alarm. For example, the following sets the description of an availability alarm for a message-queue alarm:

```
msconfig set alarm.system:diskavail.description "Percentage message-queue partition disk space available"
```

Monitoring Message Store Partitions

By default, partition monitoring is in effect so that when a message-store partition uses more than a specified percentage of available disk space, the partition is locked and any incoming messages are held in the MTA message queue. Two `msconfig` options control partition monitoring:

- `checkdiskusage`
`checkdiskusage` enables partition monitoring. It takes a boolean value; the default is 1 (monitoring is enabled).
- `diskusagethreshold`
`diskusagethreshold` specifies a disk-usage threshold beyond which the partition is locked. It takes an integer value from 1 to 99; the default value is 99.

As a partition approaches the threshold specified in `diskusagethreshold`, the message-store daemon checks the partition with increasing frequency, ranging from once every 100 minutes to once every minute. If disk usage goes higher than the threshold specified in `diskusagethreshold`, the message-store daemon:

- Locks the partition; incoming messages are held in the MTA message queue and are not

- delivered to mailboxes in the message store partition until it is unlocked.
- Logs a message to the default log file.
 - Sends an email notification to the postmaster. (You can change the recipient of the notification by setting the `msconfig alarm.noticercpt` option.)

In setting the `diskusagethreshold` option, specify a usage percentage that is low enough to allow time for repartitioning or assigning more disk space to the local message store. For example, if a partition fills up disk space at a rate of 2 percent per hour and it takes an hour to allocate additional disk space, set the disk-usage threshold to a value lower than 98 percent.

Protecting Mailboxes from Deletion or Renaming (Unified Configuration)

Protecting Mailboxes from Deletion or Renaming (Unified Configuration)

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 `store.pin` configuration option by using the following format:

```
msconfig set store.pin <mailbox1>%<mailbox2>%<mailbox3>
```

where *mailbox1*, *mailbox2*, and *mailbox3* are the mailboxes to be protected (you can use spaces in mailbox names), and `%` is the separator between each mailbox.

In this example above, the mailboxes specified in `<mailbox1>%<mailbox2>%<mailbox3>` are not per user mailboxes, as in `user/user1/Drafts`. Instead, the mailboxes specified are for all users using a certain directory, such as `Drafts`. An administrator can therefore prevent users from renaming or deleting the `Drafts` or `Backup` folder for all users by running:

```
msconfig set store.pin Drafts%Backup
```

Reducing Message Store Size Due to Duplicate Storage in Unified Configuration

Reducing Message Store Size Due to Duplicate Storage in Unified Configuration

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:

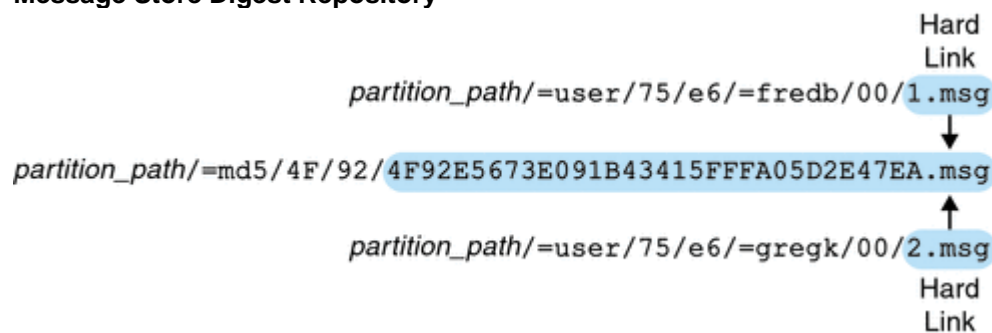
- [Relinker Theory of Operations](#)
- [Using Relinker in the Realtime Mode](#)
- [Configuring Relinker](#)

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 the topic on message store directory layout in *Messaging Server System Administrator's Guide*). Messages in the digest repository are uniquely identified by their MD5 digest. For example, if the digest for `fredb/00/1.msg` is `4F92E5673E091B43415FFFA05D2E47EA`, 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 `msconfig` option `store.relinker.enable` to 1. 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 does not 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 have 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 options used to set `relinker` criteria.

relinker msconfig options

option	Description
store.relinker.enable	<p>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.</p> <p>Default: 0</p>
store.relinker.maxage	<p>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.</p> <p>Default: 24</p>
store.relinker.minsize	<p>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.</p> <p>Default: 0</p>
store.relinker.purgecycle	<p>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).</p> <p>Default: 24</p>

Specifying Administrator Access to the Message Store in Unified Configuration

Specifying Administrator Access to the Message Store in Unified Configuration

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 in Unified Configuration](#) for conceptual information.

Topics:

- [Overview](#)
- [Adding an Administrator Entry](#)
- [Modifying or Deleting an Administrator Entry](#)

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 \(Unified Configuration\)](#).

Adding an Administrator Entry

To add an administrator entry at the command line, enter:

```
msconfig set store.admins <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`.

Modifying or Deleting an Administrator Entry

To modify or delete an existing entry in the message store Administrator UID list at the command line, use the same command:

```
msconfig set store.admins <adminlist>
```

where *adminlist* is a space-separated list of administrator IDs who should be included in the modified list. 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 can delete members from the list, but the modified list must contain at least one administrator ID.

You must restart `imapd` for the system to recognize the change in `store.admins`.

Troubleshooting the Message Store in Unified Configuration

Troubleshooting the Message Store in Unified Configuration

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:

- [Repairing Mailboxes and the Mailboxes Database \(reconstruct Command\)](#)
- [Reduced Message Store Performance](#)
- [Convergence Not Loading Mail Page](#)
- [Command Using Wildcard Pattern Does Not Work](#)
- [Unknown/invalid Partition](#)
- [User Mailbox Directory Problems](#)
- [Store Daemon Not Starting](#)
- [User Mail Not Delivered Due to Mailbox Overflow](#)
- [IMAP Events Become Slow](#)

See also:

- [Managing Logging in Unified Communications](#)
- [The topic on upgrading the message store in *Messaging Server System Administrator's Guide*](#)

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` documentation in *Messaging Server System Administrator's Guide* for more details.

Reduced Message Store Performance

Message store problems can occur if the `mbxlist` 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.
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
```

Convergence Not Loading Mail Page

If users accessing their mail with web clients, like Convergence, cannot load pages, the problem might be that the data is getting corrupted after compression. This can sometimes happen if the system has deployed an outdated proxy server. To solve this problem, try setting the `msconfig http.gzipstatic` and `http.gzipdynamic` options 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 the topic on repairing mailboxes and the mailboxes database in *Messaging Server System Administrator's Guide*.

Store Daemon Not Starting

If `stored` does not 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 the following:

```
[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.

Long-term, you will need to inform the user of mailbox size limitations, implement an aging policy (see [Configuring Message Expiration in Unified Configuration \(Tasks\)](#)), a quota policy (see [Message Store Quota \(Overview\) in Unified Configuration](#)), set a mailbox limit by setting `store.maxmessages` (see http://msg.wikidoc.info/index.php/Configutil_Reference), set up an archiving system, or make an adjustment 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 Unified Configuration](#) for ENS details.

Solution: If you want ENS notifications, verify that the ENS is enabled and configured correctly. If you do not want ENS notifications, make sure that `libibiff` is not being loaded. Typical incorrect configuration:

```
notifytarget = /opt/sun/comms/messaging/lib/libibiff
ens.enable = 0
```

Instead, use one of the following configurations:

```
notifytarget =
ens.enable = 0
```

or

```
notifytarget = /opt/sun/comms/messaging/lib/libibiff
ens.enable = 1
```

Valid Message Store UIDs and Folder Names in Unified Configuration

Valid Message Store UIDs and Folder Names in Unified Configuration

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](#)
- [Message Store Mailbox Name for Commands](#)
- [Valid UIDs](#)

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 hexadecimal (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 `ldap_uid_invalid_chars`

option 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=32,33,34,35,36,37,38,40,41,42,43,44,47,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255
```

Valid mail folder names. The following characters are invalid as folder names:

% * ? and ASCII values less than 20 or greater than 7E hexadecimal (see `man ascii`).

In addition, folder names must be valid UTF-7 sequences.

Chapter 16. Messaging Server Lemonade Profile 1 Support in Unified Configuration

Lemonade Profile 1 Support in Unified Configuration

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.

Topics:

- [Introduction to Lemonade](#)
- [Lemonade Features](#)
- [Support for BURL](#)
- [IMAP URLAUTH Support](#)
- [IMAP CATENATE Support](#)
- [IMAP Conditional Store Operation Support](#)
- [IMAP ANNOTATE Support](#)
- [Controlling IMAP CAPABILITIES Vector](#)
- [Support for SMTP Submission Service Extension for Future Message Release](#)

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 information 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 Lemonade 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 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 [BURL Support for SMTP SUBMIT in Unified Configuration](#) for details on configuring and using BURL.

Starting with version 7.0, Messaging Server 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>.

Starting in version 7, support is enabled in Messaging Server by the 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 authenticated UID. The `uid` will be blank if no authentication has been performed. The `:$` input flags will be set if SASL authentication has been performed and `:$: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.

The default BURL settings for Unified Configuration are the following:

```
BURL_ACCESS
```

```
*|tcp_*|%*|                                imap$Y  
*|*@*|imap://*;URLAUTH=submit+$1*:* $:A$M$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 `imap.submituser` option if not specified. The `imap_password` option specifies the password which of course must match the value set for the submit user account. The `imap_password` option has no default value.

IMAP URLAUTH Support

Starting with version 7.0, Messaging Server 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

Starting with version 7.0, Messaging Server 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 concatenate 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

Starting with version 7.0, Messaging Server 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

Starting with version 7.0, Messaging Server 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, it is important that the systems advertise consistent IMAP extension sets, especially with respect to CONDSTORE. During migration you can configure Messaging Server 7 to display the same capability set as the older Messaging Server version. You can also 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.

You can also filter the initial capability vector advertised in the IMAP banner in the MMP.

Set the IMAP capability options to 1 to control the CAPABILITIES vector. To see a list of these options, run the following command:

```
msconfig
msconfig> help option capability_*
```

You can also refer to the [Messaging Server Reference](#) to see the IMAP capability options.

The default values for most of these options is 1. The exceptions are IMAP4 and CONDSTORE. The default for IMAP4 is 0 unless `obsoleteimap` is set, in which case it is 1. These options only affect whether a particular feature is advertised, except for `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 are only 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

Starting with version 7.0, Messaging Server 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 that 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 option takes a single integer argument: the maximum

number of seconds a message can be held.

Chapter 17. 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
- Probe format
- Examples of LOG_ACTION use
 - Disabling logging of connections from a periodic monitoring source
 - Syslog notices after SMTP AUTH attempts with bad password
 - Syslog notices after SMTP AUTH attempts with bad username
 - Syslog notices after failing SMTP AUTH attempts, resetting after success
 - Syslog notices when time-in-queue becomes "high", ceasing after any quick delivery
 - Blocking submissions of local senders who may be spammers

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
\$< <i>string</i>	Send <i>string</i> 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
\$> <i>string</i>	Send <i>string</i> 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
```


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,$6,>=15]$NYous
\
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 18. Messaging Server Unified Configuration Command-line Utilities

Oracle Communications Messaging Server Unified Configuration Command-line Utilities

This information describes the Messaging Server `configure`, `configtoxml`, and `msconfig` commands that you use to manage a Unified Configuration. You need to be logged in either as `root` or `mailsrv` to run these commands. These commands are located by default in the `msg-svr-base/bin` directory.

See [Overview of Messaging Server Unified Configuration](#) for a description of Unified Configuration.

Unified Configuration was introduced in **Messaging Server 7 Update 5**.

Topics:

- [configure Command](#)
- [configtoxml Command](#)
- [msconfig Command](#)

configure Command

The `configure` command creates an initial runtime configuration for Messaging Server. It gives you a base working configuration from which you can make your specific customizations. The command is only meant to be run once. Subsequent running of this command overwrites the existing configuration. To modify your initial runtime configuration, use the `msconfig` command. In Messaging Server 7 Update 5, the `configure` command defaults to producing a legacy configuration (non-Unified Configuration). To produce a Unified Configuration, use the `configure --xml` command.

Syntax

```
configure [<options>]
```

Options

The following table shows the options for the `configure` command:

Options for configure Command

Option	Description
<code>--debug</code>	Provides additional debugging (primarily for LDAP operations)
<code>--help,-?</code>	Shows this help
<code>--ignoreSendmail</code>	Does not disable sendmail after configuration
<code>--ldapport=port</code>	Specifies an LDAP port, if you want to use other than port 389
<code>--ldif</code>	Does not modify LDAP directory, just writes LDIF
<code>--noldap</code>	Runs without LDAP present (statefile only)
<code>--novalidate</code>	Skips most validation of user inputs
<code>--noxml</code>	Generates legacy configuration (does not use XML-based Unified Configuration); can also be used to replace a Unified Configuration with a freshly generated legacy configuration (fresh installation of Messaging Server, not an upgrade where the <code>configtoxml</code> command was run)
<code>--saveState=filename</code>	Specifies file where state information is saved
<code>--ssl</code>	Requires SSL when configuring LDAP
<code>--state=filename</code>	Uses silent state file to configure product
<code>--version,-V</code>	Shows product version
<code>--xml</code>	Generates Unified Configuration (XML)

Examples

- To configure an initial Unified Configuration after installing Messaging Server 7 Update 5:

```
configure --xml
```

- To configure an initial legacy configuration after installing Messaging 7 Update 5:

```
configure --noxml
```



Note

A statefile can use the `XMLCONFIG=0` or `XMLCONFIG=1` option to specify a legacy or Unified Configuration, respectively. The `--xml` and `--noxml` command options override what is specified in the state file.

configtoxml Command

The `configtoxml` command converts a legacy configuration to a Unified Configuration. Run this command after first upgrading from a prior Messaging Server release to Messaging Server 7 Update 5.


Syntax

```
configtoxml [<options>]
```

Options

The following table shows the options for the `configtoxml` command:

Options for configtoxml Command

Option	Description
-32,-64	Installation is 32-bit (-32) or 64-bit (-64) Messaging Server. Default is 64-bit when the installation type cannot be inferred from the <code>SERVERROOT</code> or <code>CONFIGROOT</code> environment variables or from the location of this script.
-f --force	<p> Ignores safety checks, enables running as non-root and permits overwriting of any pre-existing Unified Configuration files.</p> <div style="background-color: #ffe6e6; padding: 10px; border: 1px solid #ccc;"> <p> Caution Using this option may result in a non-functioning configuration. The <code>restricted.cnf</code> file must always be owned by <code>root</code>.</p> </div>
-h --help	Shows this help.
-i <i>INSTANCE</i>	Inserts instance name in the generated configuration files. The default is <code>ims</code> .
-l <i>DIR</i> --location <i>DIR</i>	<p> Reads the legacy configuration files from the specified directory. The default to use is determined by the following:</p> <ol style="list-style-type: none"> 1. The <code>CONFIGROOT</code> environment variable, if defined 2. The <code>SERVERROOT/config/</code> path, if the <code>SERVERROOT</code> environment variable is defined 3. Location of <code>CONFIGROOT</code>, determined from the location of this script 4. The OS-specific default path
-n --noactive	Does not generate an active configuration and does not move the legacy configuration files to the <code>CONFIGROOT/legacy-config/</code> directory. The generated Unified Configuration files have the names <code>config.xml</code> , <code>xpass.xml</code> , and <code>restricted.cnf</code> , and are written to <code>CONFIGROOT</code> . This option cannot be used in conjunction with the <code>--output</code> or <code>--undo</code> options.
-o <i>CONFIG-FILE</i> <i>PASSWORD-FILE</i> <i>RESTRICTED-FILE</i> --output <i>CONFIG-FILE</i> <i>PASSWORD-FILE</i> <i>RESTRICTED-FILE</i>	Directs the Unified Configuration file output to the designated files. By default, the files <code>config.xml</code> , <code>xpass.xml</code> , and <code>restricted.cnf</code> are written to the <code>CONFIGROOT</code> or <code>SERVERROOT/config/</code> directory. This option cannot be used in conjunction with the <code>--noactive</code> or <code>--undo</code> options.
-r <i>ROLE</i> --role <i>ROLE</i>	Inserts the role name in the generated configuration files. The default is <code>ims</code> .
-y --yes	Pre-answer any confirmation questions with a "yes" response so that this script can be run without user intervention.
-u --undo	Removes any active Unified Configuration files and restores any legacy configuration files.

Key environment variables for this command are:

- `SERVERROOT` (The default is `/opt/sun/comms/messaging64/.`)
- `CONFIGROOT` (The default is `/opt/sun/comms/messaging64/config/.`)

Example

The following example shows the `configtoxml` converting a legacy configuration to a Unified Configuration.

```
# bin/imsimta version
Oracle Communications Messaging Server 7u5-28.12 64bit (built Nov 5
2012)
libimta.so 7u5-28.12 64bit (built 15:58:11, May 23 2012)
Using /opt/sun/comms/messaging/config/imta.cnf (not compiled)
Linux host1.example.com 2.6.39-100.5.1.el5uek #1 SMP Tue Mar 6 20:25:25
EST 2012 x86_64 x86_64 x86_64 GNU/Linux

# bin/configtoxml
WARNING: This procedure will produce an active Unified Configuration
which
    will override any existing legacy configuration.

Continue anyway [no]? yes
Creating the directory /opt/sun/comms/messaging/config/legacy-config/
Moving the processed legacy configuration files to
/opt/sun/comms/messaging/config/legacy-config/
# bin/imsimta version
Oracle Communications Messaging Server 7u5-28.12 64bit (built May Nov 5
2012)
libimta.so 7u5-28.12 64bit (built 15:58:11, Nov 5 2012)
Using /opt/sun/comms/messaging/config/config.xml (not compiled)
Linux host1.example.com 2.6.39-100.5.1.el5uek #1 SMP Tue Mar 6 20:25:25
EST 2012 x86_64 x86_64 x86_64 GNU/Linux
```

Notes on the `configtoxml` Command

- Stop Messaging Server before running the `configtoxml` command. Alternatively, use the `--noactive` switch to prevent writing out an active configuration.
- When generating an active Unified Configuration, the `configtoxml` command moves all the processed legacy configuration files to the `$CONFIGROOT/legacy-config` directory. The `--undo` option removes the Unified Configuration and restores the legacy configuration files.
- The `--undo` option leaves the Unified Configuration `restricted.cnf` password file in place.

`msconfig` Command

The `msconfig` command administers the Unified Configuration.

Syntax

You can invoke the `msconfig` command in "interactive" or "non-interactive" mode.



Caution

Only edit your Unified Configuration by running the `msconfig` command. This saves old configurations and allows for rollback. Do not hand-edit any of the Unified Configuration files. Oracle Support may occasionally edit these files to work around any issues pertaining to `msconfig`.

- To invoke in non-interactive mode:

```
msconfig <command> <option> <value>
```

See the [msconfig Commands table](#) for a list of commands.

The option syntax is *scope.optionname*, where *scope* can be multiple `.` or `:` delimited name components and *optionname* is a single name component (with no `.`). In this context, *scope* is the set of groups (and group names) used in the naming convention as described in [Unified Configuration Option Names](#). The option name determines the semantics of the option, and the scope determines the part of the product to which those semantics apply. Thus, an option with "role.base" scope applies to the entire product unless the same option is used with a more specific scope.

For example, consider the `sslnickname` option. A setting for `base.sslnickname` applies to the entire product, but a different one can be specified for different parts of the product. For example, `imap.sslnickname` only applies to the IMAP server (while the rest of the servers would use `base.sslnickname`).

The `.` delimiter appears before a name component of an option scope that is predetermined. The `:` delimiter appears before a name component of an option scope that is customer supplied and usually extensible.

The *scope* prefix is optional if unambiguous. There is also a higher-level scope of *instance* or *role* that you normally do not specify but that the `msconfig` command does display. For example:

```
role.channel:ims-ms.official_host_name
role.channel:ims-ms.backoff
role.channel:ims-ms.defragment
role.channel:ims-ms.fileinto
role.channel:ims-ms.maxjobs
role.channel:ims-ms.notices
role.channel:ims-ms.pool
```

Some scopes have associated names. For example, `schedule.task:expire.crontab` sets the `crontab` option in the `task` named `expire` in the `schedule` scope.

- To invoke in interactive mode:

```
msconfig
```

When run interactively, the default prompt for `msconfig` changes to the following:

```
msconfig>
```

After you have modified the configuration, the prompt changes to the following:

```
msconfig#
```

The `DEFAULT`, `INSTANCE`, and `ROLE` commands also affect the prompt. In default mode:

```
msconfig>
```

In Instance mode:

```
msconfig.instance>
```

In Role mode:


```
msconfig.role>
```

Options

The following table lists the commands for `msconfig`:

msconfig Commands

Command	Description
DEFAULT	Uses option's default location
DIRECTORY <i>[filter]</i>	Shows available recipes, optional <i>filter</i> matches string
DIFFERENCES <i>[m [n]]</i>	Compares configurations
EDIT <i>object</i>	Edits by using external editor
EXECUTE <i>command</i>	Executes single recipe command
EXIT	Exits <code>msconfig</code> utility with option to write
HELP <i>[topic [subtopic ...]]</i>	Shows this help
HISTORY	Lists previous saved configurations
INSTANCE	Stores options in instance
IMPORT <i>config [pass]</i>	Reads configuration from alternate file(s)
LOG	Acts as a synonym for history
QUIT	Exits <code>msconfig</code> utility immediately
REVERT <i>[n]</i>	Discards any changes and reloads configuration
ROLE	Stores options in role
RUN <i>recipe</i>	Runs specified recipe
SET <i>option [value]</i>	Sets option to the specified value
SHOW <i>option [namefilter [valuefilter]]</i>	Shows value of option, optional <i>valuefilter</i> matches string
UNSET <i>option</i>	Deletes option from the configuration
WRITE <i>[-remark=remark-string]</i>	Writes configuration changes, includes optional remark

 **Tip**
The `msconfig help` command contains extensive documentation not only about running the `msconfig` command itself, but also on other Unified Configuration topics.

The following table describes the *object* types used with the `EDIT` sub-command. The `EDIT` sub-command places the specified *object* in a file in an appropriate textual form then invokes the editor specified by the `EDITOR` shell variable. After being edited, the file is read and any changes are incorporated into the configuration.

object Types Used with the EDIT Sub-command

object Type	Description
ALIASES [<i>alias-name</i>]	MTA aliases
CHANNELS [<i>channel-name</i>]	MTA channels
CONVERSIONS	MTA conversion channel control entries
FILTER	Sieve filter
MAPPINGS [<i>mapping-name</i>]	MTA mappings
OPTION <i>option-name</i>	Specifies an option
REWRITES	MTA rewrite rule

Notes on the msconfig Command

- The `msconfig` command checks syntax and prevents potential misconfigurations due to, for example, entering channel keywords twice or using invalid (obsolete) channel options.
- Do not run the `msconfig` and `configutil` commands together.
- The `msconfig` command validates the correctness of each individual option value when it is set and the correctness of the entire configuration.
- The `msconfig` prevents writing an invalid configuration to the active configuration.
- Configuration validation is not performed by other Messaging Server processes.
- Schema errors do not prevent Messaging Server from running, however, XML syntax errors do prevent operation.
- If the value provided to the `msconfig` option contains special characters, each special character must be prefixed with the escape character `"\"` if you are using non-interactive mode. If you use `msconfig` in interactive mode to set the value that contains special characters, you do not need to prefix them with the escape character.

Example: To set the value of the `auth.searchfilter` option to:

```
( | (uid=%U)(mail=%o) )
```

you can run the following in non-interactive mode:

```
#!/msconfig set auth.searchfilter
\"\\(|\\(uid=\\%U\\)\\(mail=\\%o\\)\\)\"
```

or you can run the following in interactive mode:

```
# ./msconfig
msconfig> set auth.searchfilter "( | (uid=%U)(mail=%o) )"
msconfig# write
```

Option Name Changes in Unified Configuration

- For some `configutil` options, simply dropping the `local.` or `service.` prefix results in the Unified Configuration name but not always. Consult the `msconfig` help documentation for a list of option names.



Tip

Use the `configutil -o legacy-name -H` command to see the Unified Configuration option name.

- Two options were "restructured" rather than just renamed in Unified Configuration: `local.store.notifyplugin` and the MMP `ServiceList`.
- Most MTA `option.dat` option names are unchanged in Unified Configuration.
- Most job controller and dispatcher option names are unchanged in Unified Configuration, with a few exceptions that were misleading.

The following table some of the structural name changes for `configutil` options.

Structural Name Changes for configutil options in Unified Configuration

Legacy Option	Unified Configuration Option
<code>logfile.*.*</code>	<code>*.logfile.*</code>
<code>sasl.default.ldap.*</code>	<code>auth.*</code>
<code>sasl.default.*</code>	<code>auth.*</code>
<code>metermaid.table.*.*</code>	<code>metermaid.local_table:*.*</code>
<code>metermaid.mtaclient.*</code>	<code>metermaid_client.*</code>
<code>metermaid.config.*</code>	<code>metermaid.*</code>
<code>alarm.*.*</code>	<code>alarm.system:*.*</code>
<code>store.quotaexceededmsg;lang-*</code>	<code>message_language:*.*quotaexceededmsg</code>
<code>gen.newuserforms;lang-*</code>	<code>message_language:*.*welcomemsg</code>
<code>schedule.*</code>	<code>schedule.task:*.*crontab</code>
<code>encryption.rsa.nssslpersonalityssl</code>	<code>base.sslnicknames</code>
<code>probe.*.*</code>	<code>msprobe.probe:*.*</code>
<code>service.http.proxy.adminpass.*</code>	<code>proxy:*.*httpadminpass</code> (host-specific proxy password)
<code>local.service.proxy.adminpass.*</code>	<code>proxy:*.*imapadminpass</code> (host-specific proxy password)
<code>local.store.notifyplugin.*.*</code>	<code>notifytarget:*.*</code>
<code>service.imap.capability.*</code>	<code>imap.capability_*</code>

Using the msconfig Command in Edit Mode

You can use the `msconfig edit object` command to edit the specified object in a file in an appropriate textual form. The `msconfig edit` command invokes the editor specified by the `EDITOR` shell variable. You can then make the change, save it, exit, and your configuration is updated.

For example, suppose you want to set the `master_debug` keyword on the `tcp_local` channel. In a legacy configuration, you need to edit the `imta.cnf` file, locate the `tcp_local` channel block, add the `master_debug` keyword to it, and save the file. This process is much simplified when you use the `msconfig` command. The equivalent operation is the following command:

```
msconfig set channel:tcp_local.master_debug
```

Even if you did not know the preceding command, you could still use the `msconfig` command to perform this operation by invoking it in edit mode:

```
msconfig edit channels
```

You can also edit a single channel block by itself by running the following command:

```
msconfig edit channel tcp_local
```

The `msconfig` command provides the same editing capability for rewrites (`edit rewrites`), mappings (`edit mappings`) and aliases (`edit aliases`).

Channel-specific option files are mapped into the Unified Configuration as sub-elements of a general "options" channel option. These options appear at the bottom of each channel block when you run `edit channels`. The following example illustrates this point:

```
tcp_local identnonnumeric inner loopcheck maysaslserver maytlsserver mx \  
  pool SMTP_POOL remotehost saslswitchchannel tcp_auth smtp sourcespamfilter1  
 \  
  switchchannel  
tcp_local-daemon  
==  
trace_level=2
```

Chapter 19. Monitoring Messaging Server in Unified Configuration

Monitoring Messaging Server in Unified Configuration

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:

- [Automatic Monitoring and Restart](#)
- [Daily Monitoring Tasks](#)
- [Utilities and Tools for Monitoring](#)

See also the topics on monitoring Messaging Server, the MTA, and the message store, in *Messaging Server System Administrator's Guide*.

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 in Unified Configuration](#).

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 the topic on gathering message store counter statistics using the `counterutil` in *Messaging Server System Administrator's Guide*.

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 in Unified Configuration](#) 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 in Unified Configuration](#).

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 the `imsimta counters` documentation 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.

For more details of the various switches available, see the help documentation on `qm summarize` by running the `imsimta qm help summarize` command.

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 categories. **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
1 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 in Unified Configuration](#) 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.

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 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 `msconfig` options shown in the table below. Further information can be found in [Automatic Restart of Failed or Unresponsive Services](#).

msprobe and watcher msconfig Options

Options	Description
<code>base.autorestart.enable</code>	Enable automatic server restart. Automatically restarts failed or hung services. Default: 1
<code>base.autorestart.timeout</code>	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 <code>msprobe</code> interval (<code>schedule.task:msprobe</code>). Default: 600 seconds
<code>msprobe.probe:service.timeout</code>	Timeout for a specific server before restart. <i>service</i> can be <code>imap</code> , <code>pop</code> , <code>http</code> , <code>cert</code> , <code>job_controller</code> , <code>smtp</code> , <code>lmtp</code> , <code>mmp</code> or <code>ens</code> . Default: use <code>msprobe.timeout</code>
<code>msprobe.probe:service.warningthreshold</code>	Number of seconds of a specific server's non-response before a warning message is logged to default log file. <i>service</i> can be <code>imap</code> , <code>pop</code> , <code>http</code> , <code>cert</code> , <code>job_controller</code> , <code>smtp</code> , <code>lmtp</code> , <code>mmp</code> or <code>ens</code> . Default: Use <code>msprobe.warningthreshold</code>
<code>msprobe.warningthreshold</code>	Number of seconds of server non-response before a warning message is logged to default log file. Default: 25 secs
<code>msprobe.queuedir</code>	MTA queue directory to check if queue size exceeded threshold defined by <code>alarm.system:diskavail.thresholddirection</code> . Default: none
<code>msprobe.timeout</code>	Period of server non-response before restarting that server. See <code>schedule.task:msprobe.crontab</code> . Default: 30 seconds
<code>schedule.task:msprobe.crontab</code>	<code>msprobe</code> run schedule. A crontab style schedule string (see schedule.task:expire.enable in the Expire and Purge Log and Scheduling Parameters Table). Note that by default, this is automatically set. See Pre-defined Automatic Tasks . To disable: set <code>schedule.task:msprobe.enable</code> to 0.
<code>watcher.enable</code>	Enable watcher which monitors service failures. (IMAP, POP, HTTP, job controller, dispatcher, message store (<code>stored</code>), <code>imsched</code> , and MMP. (LMTP/SMTP servers are monitored by the dispatcher and LMTP/SMTP clients are monitored by the <code>job_controller</code> .) Logs error messages to the default log file for specific failures. Default: 1

Alarm Messages

`msprobe` can issue alarms in the form of email messages to the postmaster (see the topic on monitoring `imapd`, `popd`, and `httpd` in *Messaging Server System Administrator's Guide*, which warns 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 `msconfig` 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 `msconfig` Options

Parameter	Description (Default in parenthesis)
<code>alarm.noticehost</code>	(localhost) Machine to which you send warning messages.
<code>alarm.noticeport</code>	(25) The SMTP port to which to connect when sending alarm message.
<code>alarm.noticercpt</code>	(Postmaster@localhost) Whom to send alarm notice.
<code>alarm.noticesender</code>	(Postmaster@localhost) Address of sender the alarm.
<code>alarm.system:diskavail.description</code>	(Percentage mail partition disk space available.) Text for description field for disk availability alarm.
<code>alarm.system:diskavail.statinterval</code>	(3600) Interval in seconds between disk availability checks. Set to 0 to disable checking of disk usage.
<code>alarm.system:diskavail.threshold</code>	(10) Percentage of disk space availability below which an alarm is sent.
<code>alarm.system:diskavail.thresholddirection</code>	(-1) Specifies whether the alarm is issued when disk space availability goes below threshold (-1) or above it (1).
<code>alarm.system:diskavail.warninginterval</code>	(24). Interval in hours between subsequent repetition of disk availability alarms.
<code>alarm.system:serverresponse.description</code>	(Server response time in seconds.) Text for description field for servers response alarm.
<code>alarm.system:serverresponse.statinterval</code>	(600) Interval in seconds between server response checks. Set to 0 to disable checking of server response.
<code>alarm.system:serverresponse.threshold</code>	(10) If server response time in seconds exceeds this value, alarm issued.
<code>alarm.system:serverresponse.thresholddirection</code>	(1) Specifies whether alarm is issued when server response time is greater than (1) or less than (-1) the threshold.
<code>alarm.system:serverresponse.warninginterval</code>	(24) Interval in hours between subsequent repetition of server response alarm.

Monitoring the MTA in Unified Configuration

Monitoring the MTA in Unified Configuration

Topics:

- [Monitoring the Size of the Message Queues](#)
- [Checking for Held messages](#)
- [Monitoring Rate of Delivery Failure](#)
- [Monitoring Inbound SMTP Connections](#)
- [Monitoring the Dispatcher and Job Controller Processes](#)

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 the `imsimta qm counters` and `imsimta qm summarize` commands.

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 `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 run the following command:

```
msconfig set 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 Dispatcher configuration option `MAX_LIFE_TIME`. 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 20. MTA Address Translation and Routing in Unified Configuration

MTA Address Translation and Routing in Unified Configuration

This information describes the flow of data through the MTA using direct LDAP data access.

Topics:

- [The Direct LDAP Algorithm and Implementation](#)
- [Address Reversal](#)
- [Asynchronous LDAP Operations](#)
- [Settings Summary](#)
- [Processing Multiple Different LDAP Attributes with the Same Semantics](#)

The Direct LDAP Algorithm and Implementation

The following sections describe direct LDAP processing:

- [Domain Locality Determination](#)
- [Alias Expansion of Local Addresses](#)
- [Processing the LDAP Result](#)
- [To Modify Group Membership Attribute Syntax](#)

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.

Topics in this section:

- [Rewrite Rule Machinery](#)
- [Domain Map Determination of Domain Locality](#)
- [Caching Of Domain Locality Information](#)
- [Error Handling](#)
- [Pattern for Domain Check Rewrite Rule](#)
- [Putting It All Together](#)

Rewrite Rule Machinery

The MTA rewrite rule machinery checks a given string to see if it is a domain that needs to be locally handled. This is activated by a *\$V* or *\$Z* metacharacter. These metacharacters are syntactically similar to the *\$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 non-local 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 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 `hostname` `msconfig` option and the list of hosts specified by the `mta.ldap_host_alias_list` `msconfig` option. 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 non-local.
The handling of domains considered to be non-local 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` is 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 `ugldapbasedn` `msconfig` option. 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 `msconfig` 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 non-local.

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 `dcroot` `msconfig` option. 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 `dcroot` option 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 `dcroot` `msconfig` option. The `dcroot` option 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_domain_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, because when they occur, it is impossible to know if a given domain is local. 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 configuration needs the following rewrite rule:

```
$*      $A$E$F$U%$H$V$H@&/IMTA_HOST/
```

and the `domain_failure` MTA option needs the following value:

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

In this rewrite rule, `IMTA_HOST` 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 the configuration under normal circumstances.

The ordering here is especially tricky. The MTA checks `$V` after the address is rebuilt but before the route is added. This allows the MTA to change the route in the event of a temporary lookup failure. Pending channel match checks are applied any time the insertion point changes, so the `@` after the second `$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. 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 differs 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 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 the `imsimta test -domain` documentation in *Messaging Server Administration Reference* for more information.
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 `base.defaultdomain` `msconfig` option. If they do not match, the entry is a member of a hosted domain. The `base.defaultdomain` option can be overridden by setting the `ldap_default_domain` MTA option.
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 is ignored and the base of the user tree specified by the `ugldapbasedn` `msconfig` option is returned.

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

- `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. 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 `msconfig` option `mta.mailaliases.ldap_mail_aliases`. If this option is not set, the `mta.ldap_schematag` `msconfig` option 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 `mta.ldap_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 `mta.ldap_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. 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 following form is created and alias processing terminates successfully with this result.

```
@<smarthost>:<user>@<domain>
```

Additionally, the conversion tag for the domain obtained from the `ldap_domain_attr_conversion_tag` MTA option is, if present, 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 is 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 is now `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 `mta.ldap_schematag` option.

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

Object Classes Resulting from Various `schematag` Values

<code>schematag</code>	User Object Classes	Group Object Classes
<code>sims40</code>	<code>inetMailRouting+inetmailuser</code>	<code>inetMailRouting+inetmailgroup</code>
<code>nms41</code>	<code>mailRecipient + nsMessagingServerUser</code>	<code>mailGroup</code>
<code>ims50</code>	<code>inetLocalMailRecipient+inetmailuser</code>	<code>inetLocalMailRecipient + inetmailgroup</code>

The information in this table, like the rest of schema tag handling, is hard coded. However, there are also two MTA options, `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, 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

<code>schematag</code>	Type	General	Mail-specific
<code>sims40</code>	users	<code>inetsubscriberstatus</code>	<code>mailuserstatus</code>
<code>sims40</code>	groups	none	<code>inetmailgroupstatus</code>
<code>nms41</code>	users	none	<code>mailuserstatus</code>
<code>nms41</code>	groups	none	none
Messaging Server 5.0	users	<code>inetuserstatus</code>	<code>mailuserstatus</code>
Messaging Server 5.0	groups	none	<code>inetmailgroupstatus</code>

If necessary the `ldap_user_status` and `ldap_group_status` MTA options 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 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` (see [Maximum allowed recipients or bad](#)

commands), and `ldap_sourceblocklimit` (see [Message size limits](#)) 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 if 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 `hostname msconfig}}` option and against the list of values specified by the `{{mta.ldap_host_alias_list msconfig` option. 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:

```
mailRoutingAddress: @<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
<code>ldap_disk_quota</code>	<code>mailQuota</code>
<code>ldap_message_quota</code>	<code>mailMsgQuota</code>

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
<code>ldap_program_info</code>	<code>mailProgramDeliveryInfo</code>	<code>\$P</code>
<code>ldap_delivery_file</code>	<code>mailDeliveryFileURL</code>	<code>\$F</code>
<code>ldap_spare_1</code>	no default	<code>\$1E \$1G \$E</code>
<code>ldap_spare_2</code>	no default	<code>\$2E \$2G \$G</code>
<code>ldap_spare_3</code>	no default	<code>\$3E \$3G</code>
<code>ldap_spare_4</code>	no default	<code>\$4E \$4G</code>
<code>ldap_spare_5</code>	no default	<code>\$5E \$5G</code>

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 <code>.HELD</code> 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. 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.
\$nA	Insert the <i>_n_</i> th character of the address. The first character is character 0. The entire address is substituted if <i>n</i> is omitted. This is intended to be used to construct autoreply directory paths.
\$D	Insert the domain part of the address.
\$nE	Insert the value of the <i>_n_</i> th spare attribute. If <i>n</i> is omitted the first attribute is used.
\$F	Insert the name of the delivery file (<code>mailDeliveryFileURL</code> attribute).
\$nG	Insert the value of the <i>_n_</i> th spare attribute. If <i>n</i> is omitted the second attribute is used.
\$nH	Insert the <i>_n_</i> th component of the domain from the original address counting from 0. The default is 0 if <i>n</i> is omitted.
\$nI	Insert hosted domain associated with alias. This metacharacter accepts an integer parameter <i>n</i> whose semantics are described in Integers Controlling Behavior Modification of the \$nI and \$nS Metacharacters .
\$nJ	Insert the <i>_n_</i> th part of the host domain counting from 0. The default for <i>n</i> is 0.
\$_n_O	Insert source route associated with the current address. This metacharacter accepts an integer parameter <i>n</i> 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 <code>LDAP_UG_FILTER</code> output-only MTA option.
\$L	Insert the local part of the address.
\$nM	Insert the <i>_n_</i> th character of the UID. The first character is character 0. The entire UID is substituted if <i>n</i> is omitted.
\$P	Insert the program name (<code>mailProgramDeliveryInfo</code> attribute).
\$nS	Insert subaddress associated with the current address. This metacharacter accepts an integer parameter <i>n</i> whose semantics are described in Integers Controlling Behavior Modification of the \$nI and \$nS Metacharacters .
\$nU	Insert the <i>n</i> th 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 <i>n</i> is omitted.
\$nX	Insert the <i>_n_</i> th component of the mailhost. The entire mailhost is inserted if <i>n</i> 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 if one is not (this particular behavior is needed by the <code>ims-ms</code> 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 <code>ldap_forwarding_address</code> MTA option. This defaults to <code>mailForwardingAddress</code> .

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 are also 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 by using 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
<code>mgrpMsgRejectAction</code>	(<code>ldap_reject_action</code>) Single-valued attribute that controls what happens if any of the subsequent access checks fail. Only one value is defined: <code>TOMODERATOR</code> , which if set instructs the MTA to redirect any access failures to the moderator specified by the <code>mgrpModerator</code> attribute. The default (and any other value of this attribute) causes an error to be reported and the message rejected.
<code>mailRejectText</code>	(<code>ldap_reject_text</code>) 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 multi-valued 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 multi-valued 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.

<code>mgrpDisallowedBroadcaster</code>	(<code>ldap_cant_url</code>) 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.
<code>mgrpMsgMaxSize</code>	(<code>ldap_maximum_message_size</code>) Maximum message size in bytes that can be sent to the group. This attribute is obsolete but still supported for backwards compatibility; the new <code>mailMsgMaxBlocks</code> attribute should be used instead.
<code>mgrpAuthPassword</code>	(<code>ldap_auth_password</code>) Specifies a password needed to post to the list. The presence of a <code>mgrpAuthPassword</code> 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 <code>or_clauses</code> MTA option acts on this attribute in the same way it acts on the other access check attributes.
<code>mgrpModerator</code>	(<code>ldap_moderator_url</code>) 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 <code>ldap_reject_action</code> MTA option. If <code>ldap_reject_action</code> is set to <code>TOMODERATOR</code> , this attribute specifies the moderator address(es) the message is to be sent to should any of the access checks fail. If <code>ldap_reject_action</code> 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 <code>members</code> . Finally, subsequent group attributes listed in this table are ignored.
<code>mgrpDeliverTo</code>	(<code>ldap_group_url1</code>) List of URLs which, when expanded, provides a list of mailing list member addresses.
<code>memberURL</code>	(<code>ldap_group_url2</code>) Another list of URLs which, when expanded, provides another list of mailing list member addresses.
<code>uniqueMember</code>	(<code>ldap_group_dn</code>) 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 <code>group_dn_template</code> MTA option. The default value for this option is: <code>ldap:/// \$A??sub?mail=*\$A</code> specifies the point where the <code>uniqueMember</code> DN is inserted.
<code>mgrpRFC822MailMember</code>	(<code>ldap_group_rfc822</code>) Mail addresses of members of this list.
<code>rfc822MailMember</code>	(<code>ldap_group_rfc822</code>) <code>rfc822MailMember</code> is supported for backwards compatibility. Either <code>rfc822MailMember</code> or <code>mgrpRFC822MailMember</code> , but not both, can be used in any given group.

<code>mgrpErrorsTo</code>	<code>(ldap_errors_to)</code> Sets the envelope originator (MAIL FROM) address to whatever the attribute specifies.
<code>mgrpAddHeader</code>	<code>(ldap_add_header)</code> Turns the headers specified in the attribute into header trimming ADD options.
<code>mgrpRemoveHeader</code>	<code>(ldap_remove_header)</code> Turns the headers specified into header trimming <code>MAXLINES=-1</code> options.
<code>mgrpMsgPrefixText</code>	<code>(ldap_prefix_text)</code> Adds the specified text to the beginning of the message text, if any.
<code>mgrpMsgSuffixText</code>	<code>(ldap_suffix_text)</code> Adds the specified text to the ending of the message text, if any.
No Default	<code>(ldap_add_tag)</code> 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: `mgsmanMemberVisibility` 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 non-zero 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_substitutions` 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, it begins with a reverse URL specification of the form:

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



Note

Changing `REVERSE_URL` for any reason is discouraged.

The `$V` metacharacter has already been described in the context of alias URLs. The list of attributes to search comes from the MTA option `ldap_mail_reverses`. If this option is not set, the `mta.ldap_schematag msconfig` option is examined, and depending on its value, an appropriate set of default attributes is chosen.

The following table shows the `mta.ldap_schematag` values and the default attributes chosen.

mta.ldap_schematag Values and Attributes

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

The `$R` metacharacter is used to specify the filter. See [The \\$R Metacharacter](#) for more information.

The `$N` metacharacter 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.

As always, `mta.ldap_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.

Settings for the `ldap_use_async` MTA Option

Bit	Value	Specific Use of LDAP
0	1	<code>ldap_group_url1</code> (<code>mgrpDeliverTo</code>) URLs
1	2	<code>ldap_group_url2</code> (<code>memberURL</code>) URLs
2	4	<code>ldap_group_dn</code> (<code>UniqueMember</code>) DNS
3	8	<code>auth_list</code> , <code>moderator_list</code> , <code>sasl_auth_list</code> , and <code>sasl_moderator_list</code> nonpositional list parameter URLs
4	16	<code>cant_list</code> , <code>sasl_cant_list</code> nonpositional list parameter URLs
5	32	<code>originator_reply</code> nonpositional list parameter URLs
6	64	<code>deferred_list</code> , <code>direct_list</code> , <code>hold_list</code> , <code>nohold_list</code> nonpositional list parameter URLs
7	128	<code>username_auth_list</code> , <code>username_moderator_list</code> , <code>username_cant_list</code> nonpositional list parameter URLs
8	256	alias file list URLs
9	512	alias database list URLs
10	1024	<code>ldap_cant_url</code> (<code>mgrpDisallowedBroadcaster</code>) outer level URLs
11	2048	<code>ldap_cant_url</code> inner level URLs
12	4096	<code>ldap_auth_url</code> (<code>mgrpAllowedBroadcaster</code>) outer level URLs
13	8192	<code>ldap_auth_url</code> inner level URLs
14	16384	<code>ldap_moderator_url</code> (<code>mgrpModerator</code>) URLs
15	32768	<code>ldap_jettison_url</code> (<code>mgrpJettisonBroadcasters</code>) 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

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
```

**Note**

Beginning with Messaging Server 7 Update 5, `ldap:/// $V? $N? mail? sub? $R` is the default setting for `reverse_url`. In most instances, there's no need to explicitly set it in your configuration.

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.

Processing Multiple Different LDAP Attributes with the Same Semantics

The MTA 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_internal` 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` attributes 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 21. MTA Concepts in Unified Configuration

MTA Concepts in Unified Configuration

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

Topics:

- [The MTA Functionality](#)
- [MTA Architecture and Message Flow Overview](#)
- [The Dispatcher](#)
- [Rewrite Rules](#)
- [Channels](#)
- [The MTA Directory Information](#)
- [The Job Controller](#)

The MTA Functionality

The Message Transfer Agent, or *MTA* is a component of the Messaging Server (see the [High Level Message Store and MTA Architecture](#) figure). 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:

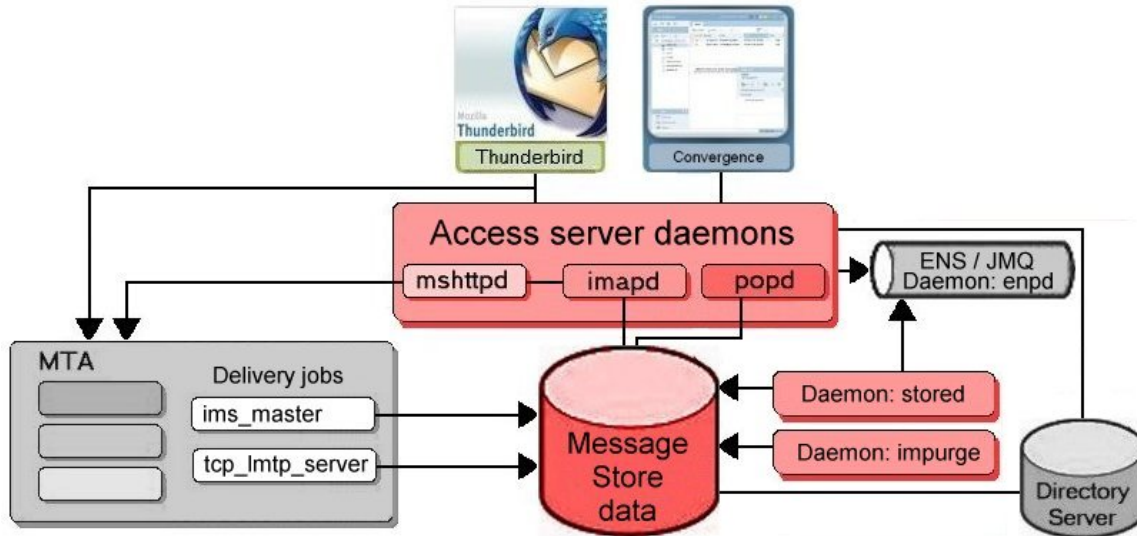
- **Routing.** Accepts a message, expands or transforms it if necessary (for example if it is an alias), and routes it to the next server, channel, program, file, or whatever. The routing function has been expanded to allow administrator specification of the internal and external mechanics of how messages 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 Unified Configuration options, mapping tables, channel options, channels, and rewrite rules. These are described in the following MTA information:

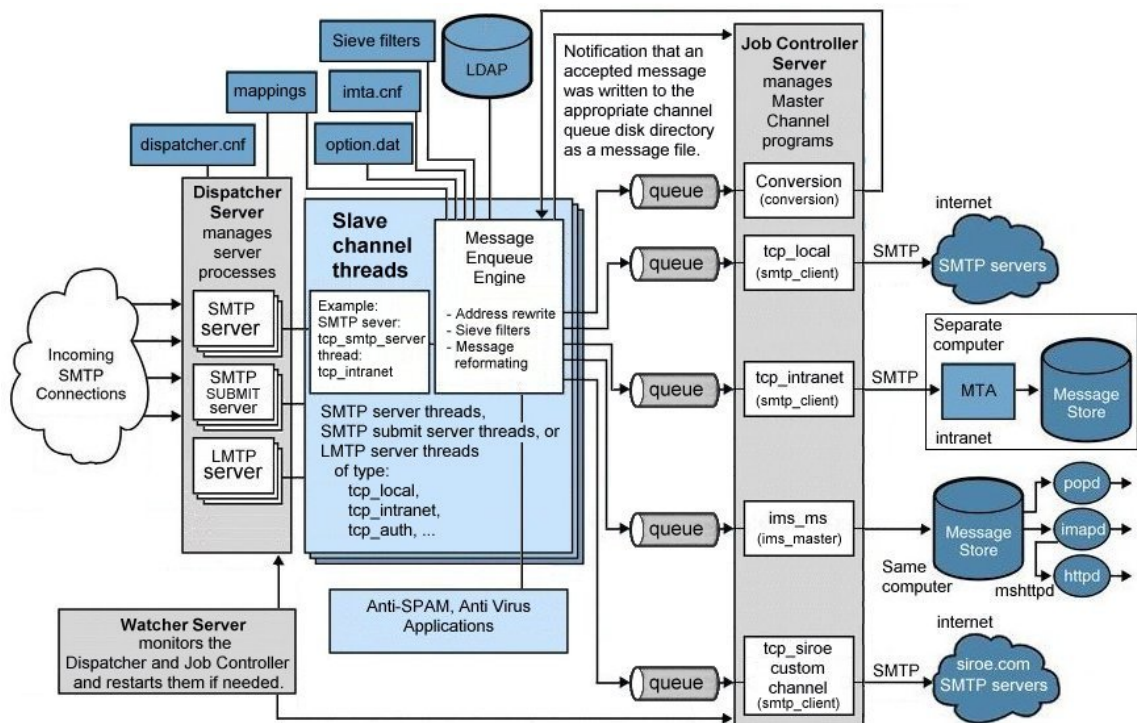
- [About MTA Services and Unified Configuration](#)
- [Channel Configuration](#)

- Configuring Rewrite Rules in Unified Configuration
- Integrating Spam and Virus Filtering Programs Into Oracle Communications Messaging Server in Unified Configuration
- LMTP Delivery in Unified Configuration
- Mail Filtering and Access Control in Unified Configuration
- Managing Logging in Unified Configuration
- Monitoring Messaging Server in Unified Configuration
- Security and Access Control in Unified Configuration
- Troubleshooting the MTA in Unified Configuration
- Using Predefined Channels in Unified Configuration
- Vacation Automatic Message Reply in Unified Configuration

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 in Unified Configuration](#)).
- **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.

Channel

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 is processed differently from mail to be delivered to the Internet, which is 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. Here is an example of a channel entry:

```
tcp_intranet smtp mx single_sys subdirs 20 noreverse maxjobs 7 SMTP_POOL
\  
maytlssserver 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. See [Channel Options Reference](#) for a complete description of channel options.

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 `service:name.min_procs` and `service:name.max_procs`. The `service:name.min_procs` option provides a guaranteed level of service by having a number of server processes ready and waiting for incoming connections. The `service:name.max_procs` option, 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. The `min_conns` option specifies the number of connections that flags a server process as "busy enough," while the `max_conns` option 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 the value of the `service:name.min_procs` option 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, run the following 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, run the following 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, run the following 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 in Unified Configuration](#) for more details.

For more information about configuring rewrite rules, see [The MTA Configuration File](#) and [Configuring Rewrite Rules in Unified Configuration](#).

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 a variety of functions, including:

- Transmitting messages to remote systems, deleting them from their queue after they are sent
- Accepting messages from remote systems, placing them in the appropriate channel queues
- Delivering messages to the local message store
- Delivering 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 by 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](#)
- [Channel Message Queues](#)
- [Channel Definitions](#)

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.

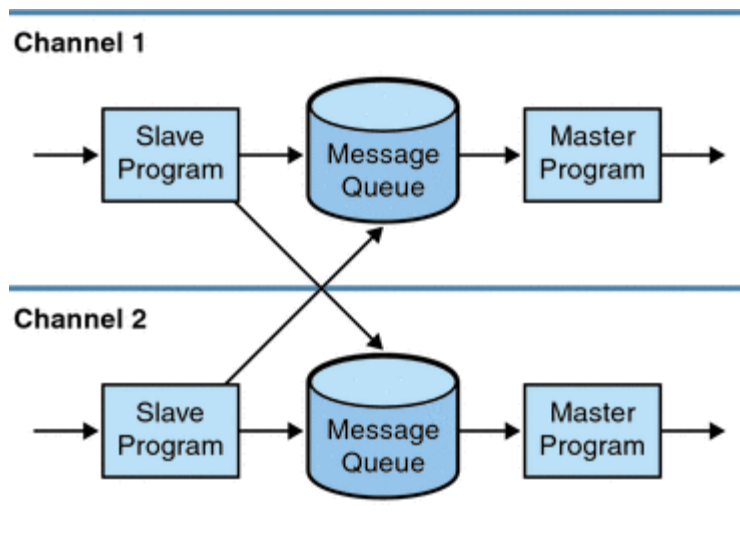
The slave channel program typically accepts incoming connections where the MTA is responding to an external request. The slave channel program:

- 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 Program Interaction](#) 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.

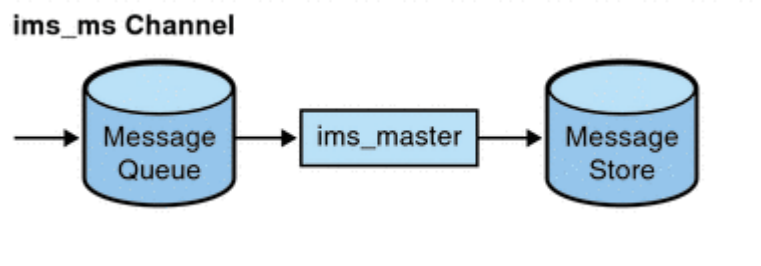
The master program dequeues the message from the queue and initiates network transport of the message. Note that the master program can only dequeue messages from its own channel queue.

Master and Slave Program Interaction



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/*`. Information on message queue sizing is discussed in the topic on disk sizing for MTA message queues in *Unified Communications Suite Deployment Planning Guide*.



Caution

Do not add any files or directories in the MTA queue directory. When using a separate file system for the MTA queue directories, create a subdirectory under that mount point and specify that subdirectory in the `SERVERROOT` environment variable. Sites may change it by using either a symbolic link or using it as a file system mount point. The default value is:

```
IMTA_ROOT:data/queue/.
```

Channel Definitions

In Unified Configuration, use the `msconfig edit channels` and `msconfig edit rewrites` commands to view and edit the channel block. (See [Configuring Rewrite Rules in Unified Configuration](#) for information about setting up channels.)

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. The following represents the channel format.

```
[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
1
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 looks something like the following, shown in legacy format as would display using the `msconfig edit channels` command:

```

tcp_intranet smtp mx single_sys subdirs 20 noreverse maxjobs 7 SMTP_POOL
\
maytlserver 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 be processed in many ways. A complete listing of channel keywords is listed and described in [Channel Options Reference](#).

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, `1`. (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 or set options for a specific channel. For more information on the option files, see [Option File](#), and [TCP/IP \(SMTP\) Channel Option Files](#). For details on configuring channels, see http://msg.wikidoc.info/index.php/Channel_configuration. For more information about creating MTA channels, see [About MTA Services and Unified Configuration](#).

- [Master and Slave Programs](#)
- [Channel Message Queues](#)
- [Channel Definitions](#)

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 in Unified Configuration](#).

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 http://msg.wikidoc.info/index.php/Channel_configuration.

Job limits for the channel are determined by the `channel:name.maxjobs` option. Job limits for the pool are determined by the `job_controller.job_pool:name.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` option. As soon as the time specified in the `backoff` option 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.

In previous versions of Messaging Server, the Job Controller read all the files in the queue directory in the order in which they are found. It now reads several channel queue directories at once. This makes for much more reasonable behavior on startup, restart, and after `max_cache_messages` has been exceeded. The number of directories 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_cache_messages` option. By increasing the `max_cache_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. 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_cache_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_cache_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_cache_messages` and the number of delayed messages is greater than 3/8 of `max_cache_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_cache_messages`.

The only obvious problems with having `max_cache_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 finds 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_cache_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_cache_messages` rounded up to a power of two. Being a hash table, this tends 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 [Recompiling the MTA Configuration](#) and http://msg.wikidoc.info/index.php/Channel_configuration.

To Start and Stop the Job Controller

To start the Job Controller, run the following command:

```
start-msg job_controller
```

To shut down the Job Controller, run the following command:

```
stop-msg job_controller
```

To restart the Job Controller, run the following 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 22. Security and Access Control in Unified Configuration

Security and Access Control in Oracle Communications Messaging Server Unified Configuration

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:

- [Configuring Authentication Mechanisms in Messaging Server Unified Configuration](#)
- [Configuring Client Access to POP, IMAP, and HTTP Services in Unified Configuration](#)
- [Configuring Encryption and Certificate-Based Authentication in Unified Configuration](#)
- [User and Group Directory Lookups Over SSL in Unified Configuration](#)

Configuring Authentication Mechanisms in Messaging Server Unified Configuration

Configuring Authentication Mechanisms in Messaging Server Unified Configuration

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 in Unified Configuration](#).

Topics:

- [Overview](#)
- [To Configure Access to Plaintext Passwords](#)
- [To Transition Users](#)

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 in Unified Configuration](#).
- **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 configuration options. For the complete listing of options, see http://msg.wikidoc.info/index.php/Configutil_Reference.

Some SASL and SASL-related Options

Parameter	Description
<code>auth.has_plain_passwords</code>	Boolean to indicate that directory stores plaintext passwords which enables APOP and CRAM-MD5. Default: 0 (False)
<code>auth.auto_transition</code>	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: 0 (False)
<code>imap.allowanonymouslogin</code>	This enables the SASL ANONYMOUS mechanism for use by IMAP. Default: 0 (False)
<code>imap pop smtp http.plaintextmncipher</code>	If this 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. The MMP has an equivalent option <code>restrictplaintextpasswords</code> . Note: <code>smtp.plaintextmncipher</code> 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
<code>auth.searchfilter</code>	This is the default search filter used to look up users when one is not specified in the <code>inetDomainSearchFilter</code> for the domain. The syntax is the same as <code>inetDomainSearchFilter</code> . (See <i>Unified Communications Suite Schema Reference</i>). Default: <code>(&(uid=%U)(objectclass=inetmailuser))</code>
<code>auth.searchfordomain</code>	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 <code>auth.searchfilter</code>) occurs directly under the LDAP tree specified by <code>base.ugldapbasedn</code> . 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.

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 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:

1. Enable the `auth.has_plain_passwords` option.

```
msconfig set auth.has_plain_passwords 1
```

2. To disable these challenge/response SASL mechanisms, disable the `auth.has_plain_passwords` option (set the value to 0).



Note

Existing users cannot use APOP and CRAM-MD5 until their password is reset or migrated (see [To Transition Users](#)).

To Transition Users

You can 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.

- Set the `auth.auto_transition` option.

```
msconfig set auth.auto_transition <value>
```

For the value, you can specify one of the following:

- 0 - Do not transition passwords. This is the default.
- 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.

Configuring Client Access to POP, IMAP, and HTTP Services in Unified Configuration

Configuring Client Access to POP, IMAP, and HTTP Services in Unified Configuration

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 (by using options such as `popproxy.domainallowed`, `imapproxy.domainallowed`, and so on). 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 Unified Configuration" 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 [Integrating Spam and Virus Filtering Programs Into Oracle Communications Messaging Server in Unified Configuration](#) 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](#)
- [Filter Syntax](#)
- [Filter Examples](#)
- [To Create Access Filters for Services](#)
- [To Create Filters by Using the Command Line](#)

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.

The following example enables multiple services on all clients.

```
+imap,pop,http:*
```

The following 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 <i>and</i> 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 `123.45.67.0/255.255.128` 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.siroel.com, .siroe.asia.com
ALL EXCEPT pop: contractor.siroel.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.siroel.com: @.siroel.com
ALL@msgServer.siroe2.com: @.siroe2.com
...
```

coupled with a Deny filter like:

```
ALL: ALL
```

Each Allow filter permits only hosts within `domainN` to connect to the service whose IP address corresponds to `msgServer.siroeN.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 by using `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 in Unified Configuration](#).

To Create Filters by Using the Command Line

You can also specify access and deny filters at the command line as follows:

- To create or edit access filters for services:

```
msconfig set <service>.domainallowed <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:

```
msconfig set <service>.domainnotallowed <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](#).

- Restart the relevant services when you make changes to their access filters.

Configuring Encryption and Certificate-Based Authentication in Unified Configuration

Configuring Encryption and Certificate-Based Authentication in Unified Configuration

This information describes encryption and certificate-based authentication for Messaging Server in a Unified Configuration.

Topics:

- [Encryption and Certificate-Based Authentication Overview](#)
- [Obtaining Certificates](#)
- [To Enable SSL and Selecting Ciphers](#)
- [Configuring Individual Messaging Processes for SSL](#)
- [Setting Up Certificate-Based Login](#)

Encryption and Certificate-Based Authentication Overview

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 and TLS version 1.0. Starting with version 7.0.5.31.0 and later, Messaging Server supports TLS 1.1 and TLS 1.2 as well.

For background information on SSL, see [Transport Layer Security](#). SSL is based on the concepts of [public-key cryptography](#).

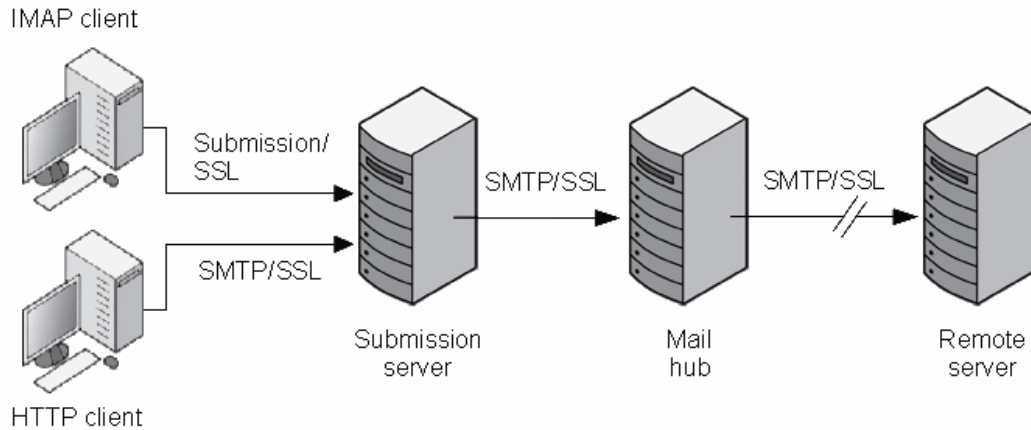
If transmission of messages between a Messaging Server and its clients and between the server and other servers is encrypted, there is reduced chance for eavesdropping on the communications. If connecting clients are authenticated, there is also reduced chance for intruders to impersonate (spoof) them.

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).

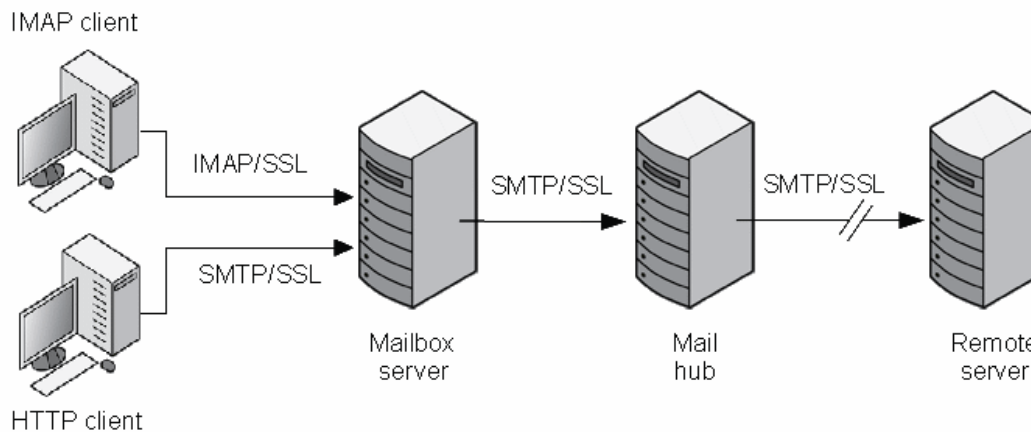
HTTP and HTTP/SSL require different ports. IMAP and IMAP/SSL, and POP and POP/SSL can use the same port or different ports. SSL acts at a specific stage of message communication, as shown in the following figure, for both outgoing and incoming messages.

Encrypted Communications with Messaging Server

A. Outgoing message



B. Incoming message



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.

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. 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 <http://www.oracle.com/technetwork/systems/archive/a11-014-crypto-accelerators-439765.pdf>.

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, for example, when using the `certutil` command.

In Unified Configuration, SSL passwords for key files are stored in the `xpass.xml` file. Use the `msconfig set -prompt "sectoken:Internal (Software) Token.tokenpass"` command to change. This command causes the `msconfig` command to prompt for the password without an echo.



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 due to recent weakness in MD5 and the NIST 2010 guidelines for SSL security strength. Use `certutil` with appropriate options (`-d` followed by a directory location with a `.sql:` prefix and `-Z SHA1 -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 the topic on SSL/TLS tasks in *Unified Communications Suite Certificate Authentication Guide*.

To Enable SSL and Selecting Ciphers

Topics in this section:

- [About Ciphers](#)
- [Specify SSL Certificate](#)

SSL certificates can be installed and configured by using the `certutil` utility and by running the appropriate `msconfig` commands 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 cryptographic 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

See the [ssladjustciphersuites](#) option for a list of available cipher suites. That 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.

The following configuration options can be used to turn on all cipher suites excluding the weak ones:

- For all services: `base.ssladjustciphersuites "all"`
- For individual services: `service.ssladjustciphersuites "all"`
where *service* is `imapproxy`, `popproxy`, or `mmp`

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

In Unified Configuration, the `.cfg` files are no longer used, and the configuration is stored in the Unified Configuration itself. In Unified Configuration, you enable and modify the configuration by running the `msconfig` command to set the appropriate options in the `imapproxy`, `popproxy`, and `smtpproxy` configuration groups. For example, in Unified Configuration, the following command sets the SSL Cipher suite for the IMAP proxy:

```
msconfig set imapproxy.ssladjustciphersuites <cipher suite>
```

Specify SSL Certificate

- To specify the SSL certificate to be presented by Messaging Server, run the following command:

```
msconfig set base.sslnicknames <certname>
```

For example:


```
# ./msconfig set base.sslnicknames "Server-Cert"
```

There is also a per-service configuration setting for the SSL server certificate nickname. The settings are as follows:

```
mta.sslnicknames for the SMTP and Submit servers
imap.sslnicknames for the IMAP server
pop.sslnicknames for the POP server
http.sslnicknames for web mail server
```

The `*.sslnicknames` options have the same meaning as (and override) the `base.sslnicknames` option. 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 options, such as `maytls`, `musttls`, and so on.

Configuring Individual Messaging Processes for SSL

This section describes the procedures you use to configure the various Messaging Server processes for SSL.

Topics in this section:

- [To Configure MMP for SSL](#)
- [To Configure IMAP for SSL](#)
- [To Configure POP for SSL](#)
- [To Configure HTTP for SSL](#)
- [To Configure SMTP for SSL](#)
- [To Verify the SSL Configuration](#)

To Configure MMP for SSL

- Use the `msconfig` command to set the following configuration parameters to enable SSL:

```
./msconfig set mmp.enable 1
./msconfig set imaproxy.tcp_listen:imaproxy1.ssl_ports 993
./msconfig set popproxy.tcp_listen:popproxy1.ssl_ports 995
```

To Configure IMAP for SSL

- Use the `msconfig` command to set the following configuration parameters to enable SSL:

```
./msconfig imap.enablesslport 1
./msconfig imap.enable 1
./msconfig imap.sslport 993
./msconfig imap.sslusessl 1
```

To Configure POP for SSL

- Use the `msconfig` command to set the following configuration parameters to enable SSL:

```
./msconfig pop.enablesslport 1
./msconfig pop.enable 1
./msconfig pop.sslport 995
./msconfig pop.sslusessl 1
```

To Configure HTTP for SSL

- Use the `msconfig` command to set the following configuration parameters to enable SSL:

```
./msconfig http.enablesslport 1
./msconfig http.enable 1
./msconfig http.sslport 443
./msconfig http.sslusessl 1
```

To Configure SMTP for SSL

1. To enable SSL encryption for outgoing messages, modify the channel definitions to include the TLS channel options such as `maytls`, `musttls`, and so on.
2. (Optional) Use the `msconfig` command to support SMTP submission with SSL on port 465 instead of the default port 587, which is enabled when a server certificate is installed and often used for SMTP submission with STARTTLS:

```
./msconfig set dispatcher.service:SMTP_SUBMIT.ssl_ports 465
```

To Verify the SSL Configuration

1. Use the `netstat` command to verify that the service is running.

```
netstat -an | grep <service>.sslport
```

where:

`service` is a keyword `mmp`, `imap`, `pop`, `http`, or `smtp`

For example:

```
# ./msconfig show 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 *msg-svr-base/log* directory. For example, check the *imap* log for 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 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 (CAs) that will issue certificates to the users that 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. Set up a certificate mapping for your server.
For example, to make the default certificate map match by the email address in the certificate subject,
make the following configuration changes at the `msconfig` prompt:

```
msconfig> set base.certmap:default.dncomps ""
msconfig# set base.certmap:default.filtercomps "e=mail"
msconfig# write
```

For more information about certificate mapping and the `dncomp` and `filtercomps` options, see [Certificate-Based Client Authentication](#).

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.

User and Group Directory Lookups Over SSL in Unified Configuration

User/Group Directory Lookups Over SSL in Unified Configuration

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 `base.ugldapport` to option to 636 enable this feature. For example:

```
msconfig set base.ugldapport 636
```

Chapter 23. Short Message Service (SMS) in Unified Configuration

Short Message Service (SMS) in Unified Configuration

This information describes how to implement the Short Message Service (SMS) in Unified Configuration on the Oracle Communications Messaging Server.

Topics:

- 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

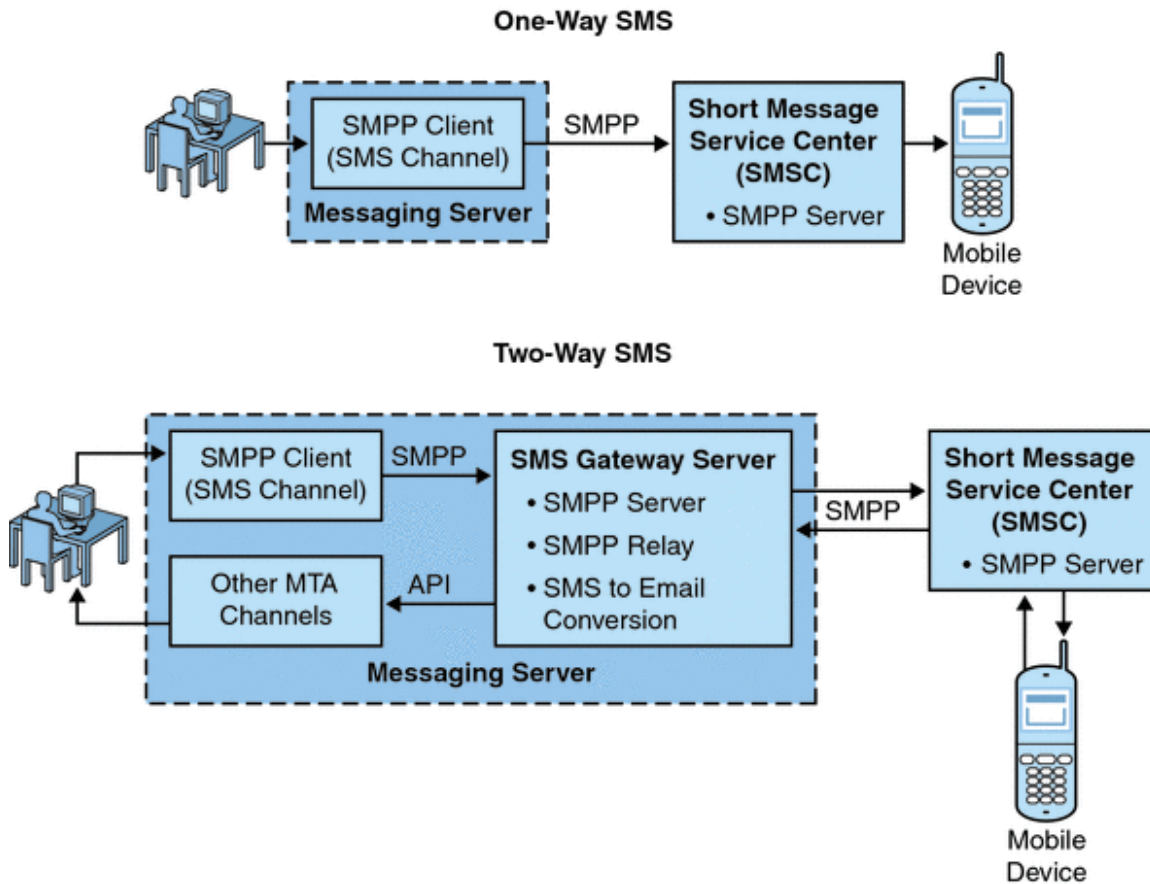
C.1 Introduction

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.](#)
- [C.2.2 The Email to SMS Conversion Process.](#)
- [C.2.3 The SMS Message Submission Process.](#)
- [C.2.4 Site-defined Address Validity Checks and Translations](#)
- [C.2.5 Site-defined Text Conversions](#)

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 <code>USE_HEADER_FROM=0</code> .
FROM_NPI	Use the specified NPI value. Ignored when option <code>USE_HEADER_FROM=0</code> .
FROM_TON	Use the specified TON value. Ignored when option <code>USE_HEADER_FROM=0</code> .
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 <code>MAXLEN</code> and the value specified by the <code>MAX_MESSAGE_SIZE</code> 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 <code>MAXPAGES</code> and the value specified by the <code>MAX_PAGES_PER_MESSAGE</code> channel option is used.
NPI	Specify a Numeric Plan Indicator (NPI) value for the destination SMS address specified with the <code>ID</code> attribute. See the description of the <code>DEFAULT_DESTINATION_NPI</code> channel option for information on the accepted values for this attribute. When this attribute is used, its value overrides the value given by the <code>DEFAULT_DESTINATION_NPI</code> 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 <code>MAX_PAGE_SIZE</code> channel option is used.
TO	Synonym for <code>ID</code> .
TO_NPI	Synonym for <code>NPI</code> .
TO_TON	Synonym for <code>TON</code> .
TON	Specify a Type of Number (TON) value for the destination SMS address given with the <code>ID</code> attribute. See the description of the <code>DEFAULT_DESTINATION_TON</code> channel option for information on the accepted values for this attribute. When this attribute is used, its value overrides the value given by the <code>DEFAULT_DESTINATION_TON</code> 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

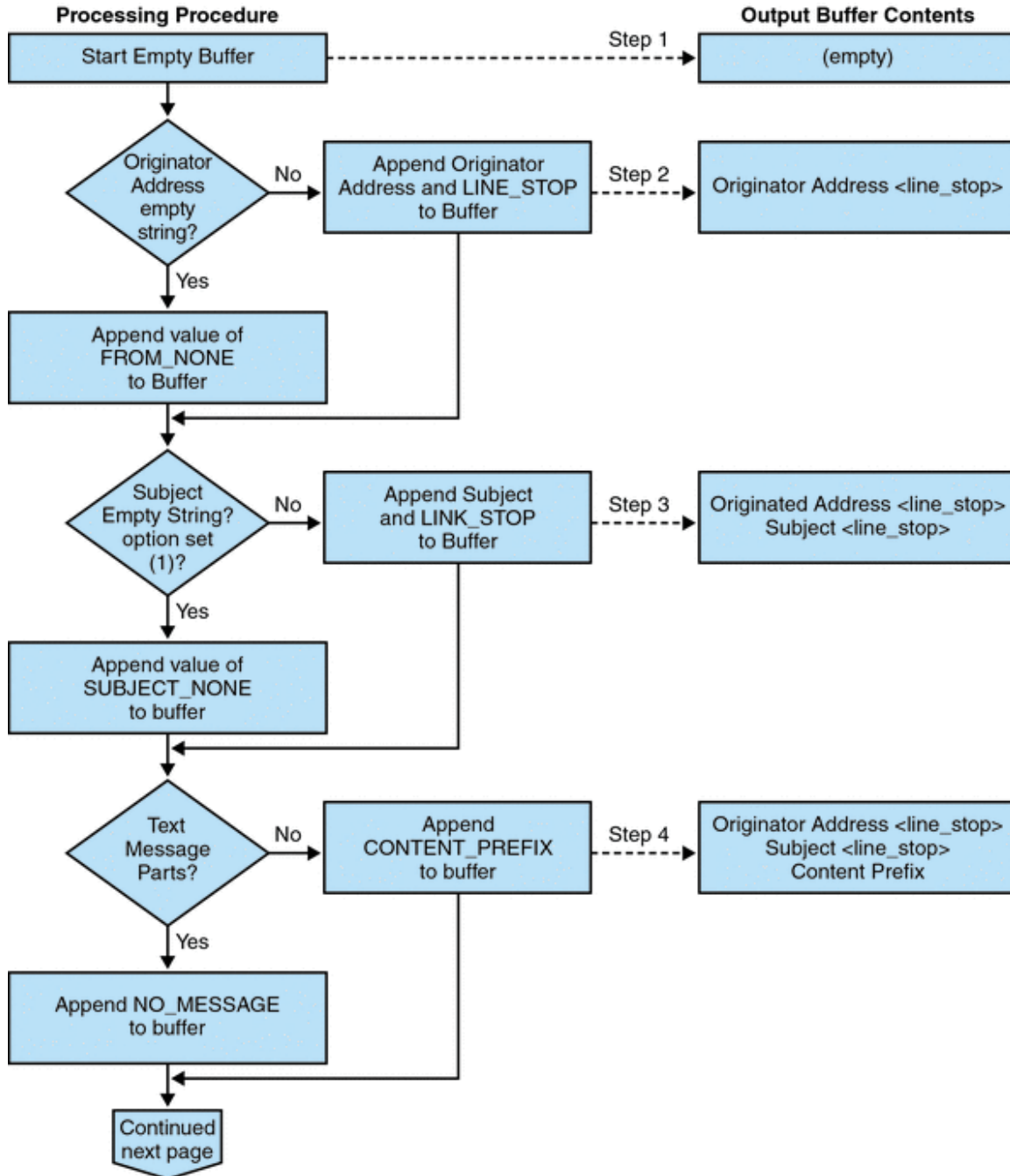
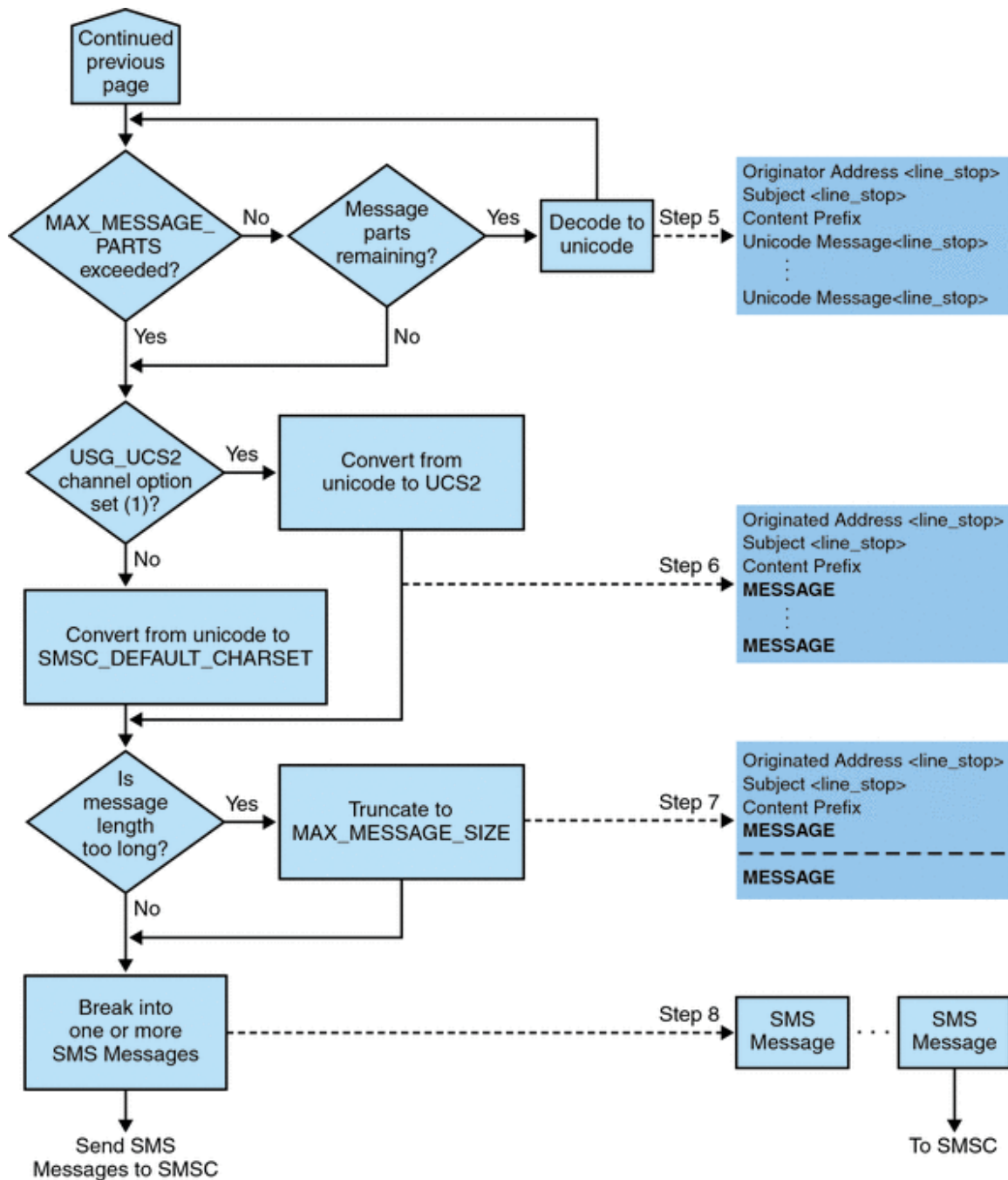


Figure C-3 SMS Channel Email Processing (continued)



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'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 option.

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
<code>system_id</code>	<code>ESME_SYSTEM_ID</code>	default value is an empty string
<code>password</code>	<code>ESME_PASSWORD</code>	default value is an empty string
<code>system_type</code>	<code>ESME_SYSTEM_TYPE</code>	default value is an empty string
<code>interface_version</code>	n/a	0x34 indicating SMPP V3.4
<code>addr_ton</code>	<code>ESME_ADDRESS_TON</code>	default value is 0x00 indicating an unknown TON
<code>addr_npi</code>	<code>ESME_ADDRESS_NPI</code>	default value is 0x00 indicating an unknown NPI
<code>addr_range</code>	<code>ESME_IP_ADDRESS</code>	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 threads 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-queued 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 option; default is to not provide this field unless the email message has a <code>Sensitivity:</code> header line
sar_refnum	See the description of the USE_SAR channel option; 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 rules:

```
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 lnnnnnnnnnnn or nnnnnnnnnnn
! In accepting it, ensure that we output +lnnnnnnnnnn
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't accept it and consequently it'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](#) using a mapping table.

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 mappings 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'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 mappings work. 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 *Messaging Server Administration Reference*.

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](#)
- [C.3.2 Setting SMS Channel Options](#)
- [C.3.3 Available Options](#)
- [C.3.4 Adding Additional SMS Channels](#)
- [C.3.5 Adjusting the Frequency of Delivery Retries](#)
- [C.3.6 Sample One-Way Configuration \(MobileWay\)](#)
- [C.3.7 Configuring the SMS Channel for Two-Way SMS](#)

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 Setting SMS Channel Options](#).

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 with channel options.

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 channel definitions or rewrite rules, you must 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, run `msconfig edit channels`. At the bottom of the channel definitions, 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 options.

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 option has been deprecated. It is ignored if present.

3. Once the channel definition has been added, run `msconfig edit rewrite` 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 changes.
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 option settings. When the `SMPP_PORT` option is used, it is not necessary to also use the `port` option. 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 option 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 Setting SMS Channel Options

In general, a channel options contain site-specific parameters required for the operation of the channel. Channel options are not required for SMS. If you determine that some are necessary for your installation, set them using `msconfig`.

For example:

```
msconfig
msconfig> set channel:sms_mway.options.profile GSM
msconfig# set channel:sms_mway.options.smsc_default_charset iso-8859-1
msconfig# set channel:sms_mway.options.use_ucs2 1
msconfig# write
```

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 <i>sar_</i> 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 <code>LISTEN_PORT</code> 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 options themselves, 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 <code>From:</code> address. Use attribute-value pair found
1	SMS source address set to <code>from-local@from-domain</code> , where the <code>From:</code> address is: <code>@from-route:from-local@from-domain</code>
2	SMS source address set to <code>from-local</code> , where the <code>From:</code> address is: <code>@from-route:from-local@from-domain</code>

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 <code>Sensitivity:</code> 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 <code>Sensitivity:</code> header line. In that case, the SMS privacy flag is set to correspond to the <code>Sensitivity:</code> 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 <code>Sensitivity:</code> header line. If the email message does not have a <code>Sensitivity:</code> 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 <i>Sensitivity</i> : 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
<i>nnn</i>	Implicit units of seconds; for example, 604800
<i>nnns</i>	Units of seconds; for example, 604800s
<i>nnnm</i>	Units of minutes; for example, 10080m
<i>nnnh</i>	Units of hours; for example, 168h
<i>nnnd</i>	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 option. 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 Options

```

msconfig
msconfig> set channel:mway_sms.options.language <default-language>
msconfig# set channel:mway_sms.options.from_prefix From:
msconfig# set channel:mway_sms.options.subject_prefix Subj:
msconfig# set channel:mway_sms.options.content_prefix Msg:
msconfig# set channel:mway_sms.options.line_stop
set channel:mway_sms.options.no_message [no message]
msconfig# set channel:mway_sms.options.reply_prefix re:
msconfig# write
msconfig> msconfig> set channel:mway_sms.options.language en
msconfig# set channel:mway_sms.options.from_prefix From:
msconfig# set channel:mway_sms.options.subject_prefix Subj:
msconfig# set channel:mway_sms.options.content_prefix Msg:
msconfig# set channel:mway_sms.options.line_stop
msconfig# set channel:mway_sms.options.no_message [no message]
msconfig# set channel:mway_sms.options.reply_prefix Re:
msconfig# write

```

```

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
<code>\$a</code>	Replace with the local and domain part of the originator's email address (for example, "user@siroe")
<code>\$d</code>	Replace with the domain part of the originator's email address (for example, "domain")
<code>\$p</code>	Replace with the phrase part, if any, of the originator's email address (for example, "Jane Doe")
<code>\$s</code>	Replace with the content of the <code>Subject:</code> header line
<code>\$u</code>	Replace with the local part of the originator's email address (for example, "user")
<code>\x</code>	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 Server 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 in Unified Configuration](#).

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 options.

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 options.

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 option 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 option 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 outbound 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 options should be set as follows:


```
msconfig
msconfig> set channel:mway_sms.options.smpp_server a.b.c.d
msconfig# set channel:mway_sms.options.smpp_port p
msconfig# set channel:mway_sms.options.esme_system_id system-id
msconfig# set channel:mway_sms.options.esme_password secret
msconfig# write
```

Now, to interoperate with MobileWay you need to make two additional option settings

```
msconfig
msconfig> set channel:mway_sms.options.esme_address_ton 0x01
msconfig# set channel:mway_sms.options.default_destination_ton 0x01
msconfig# write
```

The rewrite rule can appear as

```
sms.your-domain $u@sms.your-domain
```

And, the channel definition can appear as

```
sms_mobileway
sms.your-domain
```

Once the channel options, rewrite rules, and channel definitions 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 option 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 option, 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 set the channel debug option to 1 by running:

```
msconfig set channel:mway_sms.options.debug -1
```

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 option	Remove the <code>master</code> channel option, 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 <code>LISTEN_INTERFACE_ADDRESS</code> 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 <code>LISTEN_PORT</code> 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 options, 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_RINVMMSGID.
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).

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.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.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 option `sms_gateway.enable` must be set to the value 1. Use the following `msconfig` command to set it:

```
# msconfig set sms_gateway.enable 1
```
- To disable the gateway server, set `sms_gateway.enable` to the value 0, using the following command:

```
# msconfig sms_gateway.enable 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:

- [C.5.6.1 Configure a Mobile-to-Email Gateway Profile](#)
- [C.5.6.2 Configure a Mobile-to-Email SMPP Server](#)

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)UrgentUrgentUrgent

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 `000nnnnnnnnnnnn`.
- 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 `000nnnnnnnnnnnn` 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 (add these lines to the bottom of the the channel blocks after running `msconfig edit channels`):

```
(blank line)
sms
sms.domain.com
```

SMS Channel Options

The channel's options would then be set as follows:

```

msconfig
msconfig> set channel:mway_sms.options.smpp_server gateway.domain.com
msconfig# set channel:mway_sms.options.smpp_port 503
msconfig# set channel:mway_sms.options.use_header_from 0
msconfig# set channel:mway_sms.options.default_source_address 000
msconfig# set channel:mway_sms.options.gateway_profile sms1
msconfig# set channel:mway_sms.options.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 sms channel option settings, 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 channel options are:

```

msconfig
msconfig> set channel:mway_sms.options.FROM_FORMAT
msconfig# set channel:mway_sms.options.SUBJECT_FORMAT
msconfig# set channel:mway_sms.options.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
-----------	--------------

<p>In-memory historical record</p>	<p>Each relayed message requires $33+m+s$ bytes of virtual memory, where m is the length of the message's SMS message ID ($1 \leq m \leq 64$) and s is the length of the message's SMS source address ($1 \leq s \leq 20$).</p> <p>When <code>MAKE_SOURCE_ADDRESS_UNIQUE=0</code>, 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 <code>MAKE_SOURCE_ADDRESS_UNIQUE=0</code>].</p> <p>Note also, that the heap allocator may actually allocate larger size pieces of virtual memory for each record.</p> <p>The maximum number of records is 43 billion:</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> $(2^{32}-1)$ </div> <p>For fewer than 16.8 million records:</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> (2^{24}) </div> <p>the hash table consumes approximately 16 Mb. For fewer than 67.1 million records:</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> (2^{26}) </div> <p>the hash table consumes approximately 64 Mb; for more than 67.1 million records, the hash table consumes approximately 256 Mb.</p> <p>Double the memory consumptions for 64 bit operating systems.</p> <p>These consumptions are in addition to the memory consumption required for each record itself.</p>
<p>On-disk historical record</p>	<p>Each relayed message requires on average the following number of bytes:</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> $81+m+2s+3a+S+2i$ </div> <p>where:</p> <ul style="list-style-type: none"> • m is the average length of SMS message IDs, and $1 \leq m \leq 64$ • s is the average length of SMS source addresses, and $1 \leq s \leq 20$ • a is the average length of email addresses, and $3 \leq a \leq 129$ • S is the average length of <code>Subject</code> : header lines, and $0 \leq S \leq 80$ <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> $78+m+3a+S+2i$ </div>

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 in Unified Configuration](#)
- [Configuring Messaging Server for Two-Way SMS in Unified Configuration](#)

Configuring Messaging Server for One-Way SMS in Unified Configuration

Configuring Messaging Server for One-Way SMS in Unified Configuration

This example describes how to configure Messaging Server for one-way SMS by using Unified Configuration recipes. You could also perform the same configuration manually by using the `msconfig` command, however, using recipes is faster and makes the task repeatable. You run one recipe on the system where you are adding the SMS channelsmpp. For more information on Unified Configuration recipes, see [Recipe Language](#) and [Using Recipes](#).

Use the following [example recipe file](#) to configure Messaging Server for one-way SMS.

1. Create the source filter file (`process_sms.filter`) with the Messaging Server unified configuration.
 - a. To create a filter definition file for the source filter, enter a definition as in the following example:

```
require ["body","imap4flags"];
if body :raw :contains "Action: failed"
{
  addflag "sms";
  addflag "smserror";
}
```

- b. Save the filter definition file.
2. Make a copy of the example recipe file and save it as `recipe.rcp`.
 3. To run the recipe, type the following command:

```
cd <msg-srv-base>/sbin
./msconfig run <recipe_name>
```

4. If running a compiled configuration, recompile the MTA configuration to capture the changes you have made:

```
cd <msg-svr-base>/bin
./imsimta cnbuild
```

5. Restart Messaging Server.

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

Example Recipe to Configure One-Way SMS with Messaging Server

```
set_option("mta.enable_sieve_body", "1");

if exists_channel("sms") {
delete_channel("sms");
}
if exists_channel("process_sms") {
delete_channel("process_sms");
}

add_channel("sms", ["notificationchannel", "process_sms", "backoff",
"PT10M PT20M PT30M", "notices", "1", "official_host_name",
"sms-handle", "options.smpp_server", "127.0.0.1", "options.smpp_port",
"8500", "options.default_source_address", "000",
"options.smsc_default_charset", "UTF-16-BE", "options.esme_password",
"password", "options.esme_system_id", "smppclient"]);

add_channel("process_sms", ["sourcefilter",
"file:/opt/sun/comms/messaging64/config/process_sms.filter",
"official_host_name", "process_sms-daemon"]);

#Setting rewrite rules
append_rewrites(["sms", "$U@sms-handle"]);
```

Configuring Messaging Server for Two-Way SMS in Unified Configuration

Configuring Messaging Server for Two-Way SMS in Unified Configuration

This example describes how to configure Messaging Server for two-way SMS by using Unified Configuration recipes. You could also perform the same configuration manually by using the `msconfig` command, however, using recipes is faster and makes the task repeatable. You run one recipe on the system where you are adding the SMS channelsmpp. For more information on Unified Configuration recipes, see [Recipe Language](#) and [Using Recipes](#).

Use the following [example recipe file](#) to configure Messaging Server for two-way SMS.

1. Create the source filter file (`process_sms.filter`) with the Messaging Server unified configuration.
 - a. To create a filter definition file for the source filter, 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";
}
```

- b. Save the process definition file.
2. Make a copy of the example recipe file and save it as `recipe.rcp`.
 3. To run the recipe, type the following command:

```
cd <msg-srv-base>/sbin
./msconfig run <recipe_name>
```

4. If running a compiled configuration, recompile the Messaging Server configuration to capture the changes you have made:

```
./imsimta cnbuild
```

5. Restart Messaging Server.

```
cd <msg-svr-base>/bin
./stop-msg
./start-msg
```

Example Recipe to Configure Two-Way SMS with Messaging Server

```
set_option("mta.enable_sieve_body", "1");
set_option("sms_gateway.enable", "1");

if exists_channel("sms") {
delete_channel("sms");
}
if exists_channel("process_sms") {
delete_channel("process_sms");
}

add_channel("sms", ["notificationchannel", "process_sms", "backoff",
"PT10M PT20M PT30M", "notices", "1", "sourcefilter",
"file://opt/sun/comms/messaging64/config/sms_channel.filter",
"official_host_name", "sms-handle", "options.smpp_server", "127.0.0.1",
"options.smpp_port", "8500", "options.gateway_profile", "GATEWAY",
"options.default_source_address", "000",
"options.smsc_default_charset", "UTF-16-BE", "options.use_header_from",
"0", "options.esme_password", "password", "options.esme_system_id",
"smppclient"]);

add_channel("process_sms", ["sourcefilter",
"file://opt/sun/comms/messaging64/config/process_sms.filter",
"official_host_name", "process_sms-daemon"]);

#Setting rewrite rules
append_rewrites(["sms", "$U@sms-handle"]);

if exists_group("sms_gateway.smpp_server:SMPPSERVER") {
delete_group("sms_gateway.smpp_server:SMPPSERVER");
}

add_group("sms_gateway.smpp_server:SMPPSERVER", ["tcp_ports", "4080",
"server_host", "127.0.0.1", "server_port", "950",
"ESME_PASSWORD", "password", "ESME_SYSTEM_ID", "smppclient"]);

if exists_group("sms_gateway.smpp_relay:SMPPRELAY") {
delete_group("sms_gateway.smpp_relay:SMPPRELAY");
delete_group("sms_gateway.smpp_relay:SMPP_RELAY");
}

add_group("sms_gateway.smpp_relay:SMPPRELAY", ["server_port", "950",
"tcp_ports", "8500", "server_host", "127.0.0.1"]);

if exists_group("sms_gateway.gateway_profile:GATEWAY") {
delete_group("sms_gateway.gateway_profile:GATEWAY");
}

add_group("sms_gateway.gateway_profile:GATEWAY", ["select_re",
"([0-9]*)", "mta_channel", "sms", "smsc_default_charset", "UTF-16-BE",
"email_body_charset", "UTF-8", "email_header_charset", "UTF-8",
"text_to_subject", "1"]);
```

6. Set the JMQ parameters to create non-delivery failure notifications of any SMS messages.. For more information, see [JMQ Notification Messages and Properties in Unified Configuration](#)

(Reference).

To set the following parameters, see [Configure and Enable the Messaging Server jmqnotify Plugin](#) :

- NewMsg.enable -v 1
- jmqHost -v "127.0.0.1"
- jmqPort -v "7777"
- jmqUser -v "user"
- jmqpwd -v "xxx"
- DestinationType -v "queue"
- jmqQueue -v "ucsms1"
- Priority -v 3
- ttl -v 1000
- Persistent -v 1
- maxheadersize -v 1024
- noneinbox.enable -v 1
- msgflags.enable -v 0
- readmsg.enable -v 0

Chapter 24. SNMP Support in Unified Configuration

SNMP Support in Unified Configuration

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 this product), you can monitor certain parts of the Messaging Server. For more information on monitoring the Messaging Server refer to [Monitoring Messaging Server in Unified Configuration](#).

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](#)
- [Configuring SNMP Support for Oracle Solaris 10](#)
- [Monitoring from an SNMP Client](#)
- [SNMP Information from the Messaging Server](#)

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 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 uses 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 the second item in this list, or through the use of SNMP version 3 "context names". This allows for SNMP monitoring of Messaging Server in failover clusters.

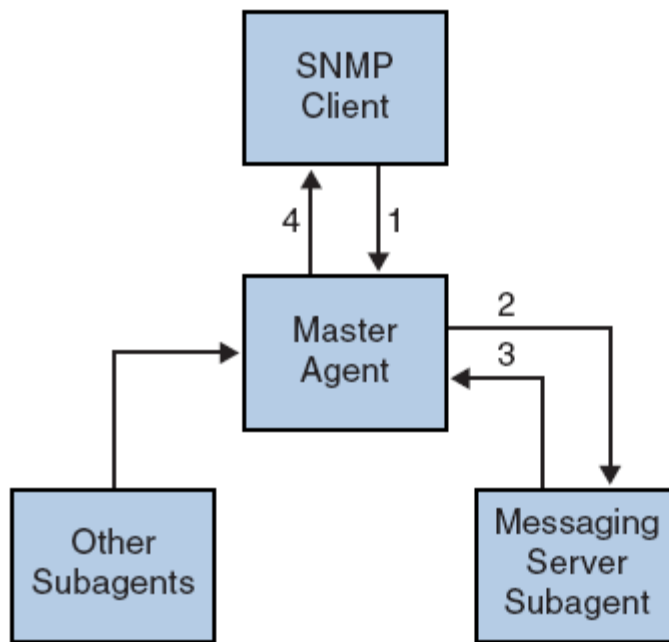
Limitations of the Messaging Server SNMP support are as follows:

- 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



1. SNMP client sends info. request to master agent
2. Master agent sends request to Messaging Server subagent
3. Messaging Server Subagent returns info to Master agent
4. Master Agent returns info to SNMP client

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 to run subagents such as Messaging Server's. 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. 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 systems, this socket by default appears as the special file `/var/agentx/master`. However, its location and name may be changed in the `snmpd.conf` 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, perform the later. This is accomplished by including the OID subtrees `mib-2.27` and `mib-2.28` in a view named "systemview" as shown in the following example. 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 are 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 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 Linux systems are as follows:

```
% ./msconfig set snmp.enable 1
% ./start-msg snmp
```

Once SNMP is running, you can test the subagent from the command line with the `snmpwalk` command. See the screen shots below for an example appropriate to Oracle Solaris and 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/sun/comms/messaging64/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/sun/comms/messaging64/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 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 to use to listen for SNMP requests. By default, it listens on all available Ethernet interfaces by using UDP port 161. In most cases, you should 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 should 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 by the `snmp.listenaddr` and `snmp.port` options.

Once you have made choices for the Ethernet interface and UDP port, set the value of the `snmp.standalone` option to one and restart the subagent. Once restarted, it operates as an 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:

```
% ./msconfig set snmp.port 9161
% ./msconfig set snmp.listenaddr 10.53.1.37
% ./msconfig set snmp.standalone 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/sun/comms/messaging64 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 `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 `snmp.port` option.

When Messaging Server's SNMP support is configured as recommended here, a monitoring station can monitor each instance of Messaging Server through 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 do 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 `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 `snmp.enablecontextname` option. If a context name different than the value of `base.defaultdomain` is desired, then set the desired name with the `snmp.contextname` option. Once each instance of Messaging Server's SNMP subagent is restarted, they can then be monitored with SNMP v3 queries that 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 in this section do not apply to the legacy SNMP subagent supplied for Oracle Solaris platforms running Oracle Solaris 9 and earlier operating systems.

SNMP Subagent Options

Option (Default)	Description
<code>snmp.enable</code> (0)	The Messaging Server SNMP subagent only runs when this option is given a value of 1, in which case Messaging Server automatically stops and starts 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 .
<code>snmp.standalone</code> (0)	Messaging Server's SNMP support normally runs as a SNMP subagent, receiving SNMP requests through the platform's SNMP master agent, <code>snmpd</code> . This operational mode is the default and is selected by giving this option a value of 0. 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 <code>snmpd</code> . 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 <code>snmp.listenaddr</code> and <code>snmp.port</code> options. To run in this standalone mode, specify a value of 1 for this option. Running in standalone mode does not interfere with other SNMP master or subagents running on the system.
<code>snmp.listenaddr</code> (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 <code>INADDR_ANY</code> . 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 <code>snmp.standalone</code> is set to 0.

snmp.cachettl (30)	<p>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.</p> <p>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 causes the subagent to obtain fresh data from Messaging Server. That is, data from Messaging Server is obtained only once every five minutes. If, on the other hand, SNMP requests are made every 10 seconds, then the subagent responds to some of those requests with cached data as old as 29 seconds. Messaging Server is polled only once every 30 seconds.</p>
snmp.servertimeout (5)	<p>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 waits for a response to each step in the protocol exchange. By default, a timeout value of five seconds is used.</p>
snmp.directoryscan (1)	<p>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, 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.</p>
snmp.enablecontextname (0)	<p>The subagent has the ability to register its MIBs under an SNMP v3 <i>context name</i>. When this is done, the MIBs may only be requested by a SNMP v3 client that 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.</p> <p>To enable the use of SNMP v3 context names, specify a value of 1 for this option. When that is done, the subagent defaults to using the value of the base.defaultdomain option for its context name. To use a different value for the context name, use the snmp.contextname option.</p>
snmp.contextname (base.defaultdomain)	<p>When the use of SNMP v3 context names has been enabled with snmp.enablecontextname, this option can 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 snmp.enablecontextname has the value 0.</p>

Monitoring from an SNMP Client

The base OIDs for [RFC 2788](http://www.faqs.org/rfcs/rfc2788.html) (<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.

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.

Topics in this section:

- [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) (<http://www.faqs.org/rfcs/rfc2788.html>) and [RFC 2789](http://www.faqs.org/rfcs/rfc2789.html) (<http://www.faqs.org/rfcs/rfc2789.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`).

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 has different thresholds and significant monitoring values. A good SNMP client allows 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.

Here 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 by 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 has 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. Here 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. `applTCPProtoID` 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`. Here 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 `app1Table`. In this example, `.1` indicates this data is for the first application in the `app1Table`. 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` gives a count of infected messages in addition to other conversion failures.

mtaGroupTable

This two-dimensional table provides channel information for each MTA in the `app1Table`. 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.

Here 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 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`.

Here 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.

Here 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 25. Troubleshooting the MTA in Unified Configuration

Troubleshooting the MTA in Unified Configuration

This information describes common tools, methods, and procedures for troubleshooting the Message Transfer Agent (MTA) in Unified Configuration.

Topics:

- [Troubleshooting Overview](#)
- [Standard MTA Troubleshooting Procedures](#)
- [Common MTA Problems and Solutions](#)
- [General Error Messages](#)

A related topic, monitoring procedures can be found in [Monitoring Messaging Server in Unified Configuration](#).



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 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.

- [Check the MTA Configuration](#)
- [Check the Message Queue Directories](#)
- [Check the Ownership of Critical Files](#)
- [Check that the Job Controller and Dispatcher are Running](#)

- [Check the Log Files](#)
- [Run a Channel Program Manually](#)
- [Starting and Stopping Individual Channels](#)
- [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 Message Transfer Agent command-line utilities documentation 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 the `imsimta qm` command and `imsimta qm counters` in *Messaging Server Administration Reference*.

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 [Managing 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:

```
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
```

Check that the files in `msg-svr-base/data/queue` are owned by the MTA account by using a command such as the following UNIX system example:

```
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:

```
# imsimta process
USER PID S VSZ RSS STIME TIME COMMAND
mailsrv2 308 S 50168 26896 23:05:00 00:01
/opt/sun/comms/messaging64/lib/tcp_smtp_server
mailsrv2 4187 S 46880 17704 Feb_17 01:51
/opt/sun/comms/messaging64/lib/dispatcher
mailsrv2 5887 S 50160 26976 23:25:00 00:00
/opt/sun/comms/messaging64/lib/tcp_smtp_server
mailsrv2 19018 S 47008 21640 Jun_20 01:21
/opt/sun/comms/messaging64/lib/job_controller
```

If the Job Controller is not present, the files in the `msg-svr-base/data/queue` directory get backed up and messages are not delivered. If you do not have a Dispatcher, then you are unable to receive any SMTP connections.

See the `imsimta process` documentation in *Messaging Server Administration Reference* for more information.

You could also use the `imsimta qm jobs` command 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
<code>channel_master.log-uniqueid</code>	Output of master program (usually client) for <i>channel</i> .
<code>channel_slave.log-uniqueid</code>	Output of slave program (usually server) for <i>channel</i> .
<code>dispatcher.log-uniqueid</code>	Dispatcher debugging. This log is created regardless if the Dispatcher <code>DEBUG</code> option is set. However, to get detailed debugging information, you should set the <code>DEBUG</code> option to a non-zero value.
<code>imta</code>	<code>ims-ms</code> channel error messages when there is a problem in delivery.
<code>job_controller.log-uniqueid</code>	Job controller logging. This log is created regardless if the Job Controller <code>DEBUG</code> option is set. However, to get detailed debugging information, you should set the <code>DEBUG</code> option to a non-zero value.
<code>tcp_smtp_server.log-uniqueid</code>	Debugging for the <code>tcp_smtp_server</code> . The information in this log is specific to the server, not to messages.
<code>return.log-uniqueid</code>	Debug output for the periodic MTA message bouncer job; this log file is created if the <code>return_debug</code> MTA option is set.

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. However, by default this command is run on a regular basis (see [Pre-defined Automatic Tasks](#)). For more information, see the `imsimta purge` documentation in *Messaging Server Administration Reference*.

The `channelmaster.log-uniqueid` and `channelslave.log-uniqueid` log files are created in any of the following situations:

- There are errors in your current configuration.
- The `master_debug` or `slave_debug` options are set on the channel.
- If `mm_debug` is set to a non-zero value (`mm_debug > 0`).

For more information on debugging channel master and slave programs, see *Messaging Server Administration Reference*.

Running 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, and 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 the `imsimta qm` documentation in *Messaging Server Administration Reference* for more information.

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 commands:

```
msconfig set
job_controller.job_pool:DEFAULT.urgent_delivery 08:00-20:00
msconfig set
job_controller.job_pool:DEFAULT.normal_delivery 08:00-20:00
msconfig set
job_controller.job_pool:DEFAULT.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 by running the `msconfig edit mapping` command:

```
ORIG_SEND_ACCESS

*|*@sesta.com|*|* $X4.2.1|$NHost$ temporarily$ blocked
```

By using this process, the sender's remote MTA holds 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 by running the `msconfig edit mapping` command:

```
PORT_ACCESS
TCP|*|25|_IP_address_to_block_|* $N400$ 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 enables 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` options to the appropriate channels. These options 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. Set the `{log_message_id}` MTA option to 1 by running the `msconfig set log_message_id 1` command. With this option set, 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` MTA option to 5 by running the `msconfig set mm_debug 5` command to provide substantial debugging information.
2. Add the `slave_debug` and `master_debug` options to the appropriate channels by running the `msconfig edit channels` command and modifying the appropriate channel definitions.
 - a. Use the `slave_debug` option 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` option is added to the `tcp_local` channel.
 - b. Add the `master_debug` option to the other channels that the message passed through and were identified in [Identify the Channels in the Message Path](#). In this example, it would be added to the `conversion` and `tcp_intranet` channels.
 - c. Recompile the configuration by using `imsimta cnbuild` if running a compiled configuration.
 - d. 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 channel options, 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 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` option 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 7](#) should correlate to the message files that you copied in [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` options from the appropriate channels by running the `msconfig channels` command.
 - b. Reset the `mm_debug` MTA option and remove the setting for the `log_message_id` MTA option by running the `msconfig set mm_debug 0` and `msconfig set log_message_id 0` commands or by running the `msconfig unset mm_debug` and `msconfig unset log_message_id` commands.
 - c. Recompile the configuration by using `imsimta cnbuild` if running a compiled configuration.
 - 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 messageAll 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 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.

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 `mailsrv`.

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` option 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` option 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 sesta.com (Step 1)

Server: LOCALHOST
Address: 127.0.0.1

Non-authoritative answer:
sesta.com preference = 10, mail exchanger = mailhost.sesta.com (Step 2)

% telnet mailhost.sesta.com 25 (Step 3)
Trying... [10.1.1.1]
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 nameserver and 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` channel option) 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` option. Channels are created in by running the `msconfig edit channels` command. The channel should have the same channel options 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 option-value pair `pool SMTP_SIROE`. This specifies that messages to this channel will only use computer resources from the `SMTP_SIROE` pool. There is a blank line before and after the new channel.

2. Add two rewrite rules by running the `msconfig edit rewrite` 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`. Run the `msconfig set job_controller.job_pool:SMTP_SIROE.job_limit 10` command to create a new pool called `SMTP_SIROE` with a `job_limit` of 10. You can verify the addition of the new pool by running the `msconfig show job_controller.job_pool` command which will show output similar to the following:

```
# msconfig show job_controller.job_pool
role.job_controller.job_pool:DEFAULT.job_limit = 10
role.job_controller.job_pool:DEFAULT.urgent_delivery = help
role.job_controller.job_pool:IMS_POOL.job_limit = 2
role.job_controller.job_pool:SMTP_POOL.job_limit = 10
role.job_controller.job_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. See [The Job Controller](#) for details on jobs and pools.

1. 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 occurrence.)

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` options) 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` configuration by running the `msconfig set job_controller.synch_time timeperiod` command 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 running the following command:

```
# ./msconfig set job_controller.synch_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 sidlined 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 option enabled in your MTA configuration 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
ZZ0HZ7003EQQSE37.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 suspicious threshold 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 option in [Expansion of Multiple Addresses](#)), 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 thresholds for determining that a message is looping are configurable; see the `MAX_*RECEIVED_LINES` MTA 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` MTA option to generate a syslog notice whenever a message is forced into .HELD state due to exceeding such a threshold. 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` command **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 is not likely to continue 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 the `clean` documentation 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 option ([Expansion of Multiple Addresses](#)), any use of the `$H` flag in address-based `*_ACCESS` mapping tables, or any use of the `hold` action in any system Sieve file, or any channel level Sieve filters configured and named by use of `sourcefilter` or `destinationfilter` channel options. See [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](#) 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` option 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 set `mm_debug` to 5 by running the `msconfig set mm_debug 5` command 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 the topic on disk sizing for MTA message queues in *Unified Communications Suite Deployment Planning Guide* 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 in Unified Configuration*, and *About MTA Services and Unified 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.

Check your MTA configuration for any channel definitions with duplicate official host names.

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.



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.

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 the `imsimta chbuild` documentation 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 the `imsimta chbuild` documentation 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).

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 [Channel configuration](#) 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.

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.

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:

`swap -s` (at the time MTA processes are busy), `ps -elf`, or `tail /var/adm/messages`

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 the `imsimta test` documentation in *Messaging Server Administration Reference* for more information.

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 rewrite rules or other configurations 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](#) [TCP/IP nameserver and MX record support](#) 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 26. Tuning the mboxlist Database Cache in Unified Configuration

Tuning the mboxlist Database Cache in Unified Configuration

Topics:

- [Setting the Mailbox Database Cache Size](#)
- [To Adjust the Mailbox Database Cache Size](#)
- [Transaction Checkpoint \(txn_checkpoint\) Warnings](#)
- [To Monitor the Mailbox Database Cache Size](#)

Setting the Mailbox Database Cache Size

Messaging Server makes frequent calls to the mailbox database. For this reason, it helps if this data is returned as quickly as possible. A portion of the mailbox database is cached to improve Message Store performance. Setting the optimal cache size can make a big difference in overall Message Store performance. You set the size of the cache with the `store.dbcachesize` option.

You should use the `store.dbtmpdir` option to redefine the location of the mailbox temporary files to a `tmpfs`, that is, `/tmp/msgDBtmpdir`. On Linux, the `tmpfs` location is usually `/dev/shm`. The default `store.dbtmpdir` value (`/tmp/.xxx`) is not appropriate, so you should change it manually to use the `tmpfs` location. On reboot, the `tmpfs` will be cleared, therefore be sure that permissions are stored so that the correct directory location can be recreated.



Note

Starting with Messaging Server 7 Update 5, the defaults for the `store.dbtmpdir` option and the `local.lockdir` option changed to `/tmp/encodedsubdirectory/store` and `/tmp/encodedsubdirectory/lockm` respectively. The `encodedsubdirectory` is `.mailuserMSinstall-location`, where `mailuser` is the Message Server `mailsrv` user and `MSinstall-location` is the install location of Messaging with slashes ("/") replaced by underscores ("_").

For example, if the `mailsrv` user is `mailuser` and Messaging Server is installed in the `/opt/sun/comms/messaging64` directory, then `/tmp/encodedsubdirectory` is `/tmp/.mailuser_opt_sun_comms_messaging64/`.

Because of these changes to the `store.dbtmpdir` and `local.lockdir` defaults, you no longer need to set these options on a Solaris platform.

To define the location of the mailbox temporary files:

1. Create the directory, for example:

```
mkdir /tmp/msgDBtmpdir
```

2. Assign the appropriate ownership and permissions so that this directory is owned, readable, and

writable by the `mailsrv` user. For example, if the `mailsrv` user name is `mailuser` and the group name is `mail`:

```
chown mailuser:mail /tmp/msgDBtmpdir
chmod 700 /tmp/msgDBtmpdir
```

3. Set the `store.dbtmpdir` option, for example:

```
# ./msconfig
msconfig> set store.dbtmpdir /tmp/msgDBtmpdir
msconfig# write
msconfig> exit
#
```

Note

Be sure to set the `store.dbtmpdir` option to a uniquely named subdirectory of a `tmpfs` file system such as `/tmp`. In Oracle Solaris Cluster environments, or any situation where it could be possible for multiple instances of Messaging Server to be running on the same system, it is essential that you set this to a name that is unique to that instance. You need to make sure that no two Messaging Server instances can ever try to use the same `tmpdir` directory on the same system, for example, `/tmp/msg-instance-dbtmp` rather than just `/tmp/mboxlist`.

The files stored in the `store.dbtmpdir` location are temporarily memory mapped files used by all processes connecting to the database. Due to their usage pattern, the pages of these files will most likely be in memory all the time. So setting this to be on a `tmpfs` will not really increase memory usage. What it does is save I/O. When the Oracle Solaris virtual memory system sees a memory mapped file on a `tmpfs`, it knows it does not really need to write the modified pages back to the file. So there is only one copy in memory and it saves I/O.

The mailbox database is stored in data pages. When the various daemons make calls to the database (`stored`, `imapd`, `popd`), the system checks to see if the desired page is stored in the cache. If it is, the data is passed to the daemon. If not, the system reads the desired page and writes it in the cache. If there is no clean page available, the system must write one page from the cache back to disk.

Lowering the number of disk read/writes helps performance, so setting the cache to its optimal size is important:

- If the cache is too small, the desired data will have to be retrieved from disk more frequently than necessary.
- If the cache is too large, dynamic memory (RAM) is wasted, and it takes longer to synchronize the disk to the cache.

Of these two situations, a cache that is too small will degrade performance more than a cache that is too large.

Cache efficiency is measured by **hit rate**. Hit rate is the percentage of times that a database call can be handled by cache. An optimally sized cache will have a 98 to 99 percent hit rate (that is, 98 to 99 percent of the desired database pages will be returned to the daemon without having to grab pages from the disk). The goal is to set the smallest cache so that it holds a number of pages such that the cache will be able to return at least 98 to 99 percent of the requested data. If the direct cache return is less than 98 percent, then you need to increase the cache size.

To Adjust the Mailbox Database Cache Size

- Use the `msconfig` command to set the size of the cache with the `store.dbcachesize` option, for example:

```
./msconfig
msconfig> set store.dbcachesize 25165825
msconfig# write
msconfig> exit
#
```

It is important to tune the cache size to smallest size that will accomplish the desired hit rate.

The `store.dbcachesize` option controls the size of a shared memory segment used by all processes connected to the database, including `stored`, `imap`, `popd`, `imsbackup`, `imsrestore`, `ims_master`, `tcp_lmtp_server`, and so on. While the maximum value for `store.dbcachesize` is 2 GB, setting it to the maximum wastes memory. Instead, start with the default value of 64 MB and monitor the cache hit rate over a period of days. Increase the value only if the hit rate is under 98%.

Also consider the transaction checkpoint function (performed by `stored`). Set the `store.checkpoint.debug` option and refresh `stored` to see log messages to provide more exact data about transaction checkpoint function time. For example:

```
./msconfig
msconfig> set -restricted store.checkpoint.debug 1
msconfig# write
msconfig> exit
# ./refresh store
Refreshing store server 7585 ... done
```

This process must examine all buffers in the cache and hold a region lock during the checkpoint. Other threads needing the lock must wait. See [Transaction Checkpoint \(txn_checkpoint\) Warnings](#) for more information.

Transaction Checkpoint (txn_checkpoint) Warnings

What Do These Warnings in the Default Log Mean?

```
[<date/time>] <hostname> stored[<pid>]: General Warning: txn_checkpoint took
84 seconds
[<date/time>] <hostname> stored[<pid>]: General Warning: txn_checkpoint took
145 seconds
[<date/time>] <hostname> stored[<pid>]: General Warning: txn_checkpoint took
107 seconds
```

One thread in the `stored` process runs database maintenance tasks every 15 seconds. Every fourth iteration (that is, once per minute), the thread runs the `txn_checkpoint` function, which commits the changes from the transaction log files to the actual database files. This is an I/O intensive operation. How long it takes depends on how many transactions have been completed since the last time it ran, the I/O rate capability, and load on the file system and storage containing the `mbxlist` directory, as well as the

overall system load. In general, expect `txn_checkpoint` to run once per minute and take less than 15 seconds.

A transaction log file is obsolete when all the transactions in that log have been committed to the database files. Prior to Messaging Server 6.3, obsolete logs were removed every five minutes. Starting in Messaging Server 6.3, they are removed once per minute.

Starting in Messaging Server 7.0, the duration of the `txn_checkpoint` function is used as a general indicator of the health of the system. These warnings are reported when it takes longer than 30 seconds. When it takes longer than 60 seconds, the store is in *stress* mode.

Running dtrace to Examine txn_checkpoint

The following `dtrace` script reports how long the `txn_checkpoint` function is taking and where it is spending the time:

```
#!/usr/sbin/dtrace -qs

/*
** txnckpt.d x0.2
**
** Usage: ./txnckpt.d -p <pid-of-stored>
**
*/

pid$target::mboxlist_txn_checkpoint:entry
{
    self->ts = timestamp;
}

pid$target::mboxlist_txn_checkpoint:return
/self->ts/
{
    this->duration = timestamp - self->ts;
    @ckptmin = min(this->duration);
    @ckptavg = avg(this->duration);
    @ckptmax = max(this->duration);
    @ckpcnt = count();

    printf("%Y txn_checkpoint took %10d nanoseconds\n",
        walltimestamp, this->duration);
    self->ts = 0;
}

syscall::*:entry
/self->ts/
{
    self->syscallstart = timestamp;
}

syscall::*:return
/self->syscallstart/
{
    this->deltaT = timestamp - self->syscallstart;
    @syscalltimes[probefunc] = quantize(this->deltaT);
    @syscalltotal = sum(this->deltaT);
}
```

```

        @syscallcount[probefunc] = count();
        @syscalltimeper[probefunc] = sum(this->deltaT);
        self->syscallstart = 0;
    }

END
{
    normalize(@ckptmin, 1000000);
    normalize(@ckptavg, 1000000);
    normalize(@ckptmax, 1000000);
    printa("txn_checkpoint ran %@d times,", @ckpct);
    printa(" duration (ms) min:%@d ", @ckptmin);
    printa(" max:%@d ", @ckptmax);
    printa(" avg:%@d \n\n", @ckptavg);

    normalize(@syscalltotal, 1000000);
    normalize(@syscalltimeper, 1000000);
    printa("%20s called %5@d times, total %6@d ms\n",
        @syscallcount, @syscalltimeper);
    printa("Total time in syscalls %30@d ms\n", @syscalltotal);

    printf("\nTime distributions for syscalls during txn_checkpoint (in

```

```
nanoseconds):\n");  
}
```

On older versions of Oracle Solaris 10, `dtrace printa` might not be able to combine aggregations as shown here. If you get an error like this:

```
dtrace: failed to compile script ./txnckpt.d: line 51: printa( )  
prototype mismatch: 5 args passed, 2 expected
```

change:

```
printa("%20s called %5@d times, total %6@d ms\n",  
      @syscallcount, @syscalltimeper);
```

to:

```
printf("individual syscall counts:");  
printa(@syscallcount);  
printf("individual syscall times (in milliseconds):");  
printa(@syscalltimeper);
```

It will produce one line of output each time `txn_checkpoint` runs. Let it run for a few minutes, then press `Ctrl-C`, and it will produce a summary report. For example:

```

# ps -ef|grep store
mailsrv 17558 2639 0 Apr 06 ? 9:25
/opt/sun/comms/messaging64/lib/stored

# ./txnckpt.d -p 17558
2009 Apr 27 10:58:58 txn_checkpoint took 63940200 nanoseconds
^C
txn_checkpoint ran 1 times, duration (ms) min:63 max:63 avg:63

           gtime called      3 times, total      0 ms
           pwrite called     4 times, total     21 ms
           fdsync called     6 times, total     37 ms
Total time in syscalls                               59 ms

Time distributions for syscalls during txn_checkpoint (in nanoseconds):

gtime
value  ----- Distribution ----- count
 1024  |                                     0
 2048  | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 2
 4096  | @@@@@@@@@@@@@@@@@@ 1
 8192  |                                     0

pwrite
value  ----- Distribution ----- count
16384  |                                     0
32768  | @@@@@@@@@@@@@@ 1
65536  |                                     0
131072 | @@@@@@@@@@@@@@ 1
262144 |                                     0
524288 |                                     0
1048576 |                                     0
2097152 |                                     0
4194304 |                                     0
8388608 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 2
16777216 |                                     0

fdsync
value  ----- Distribution ----- count
 4096  |                                     0
 8192  | @@@@@@@@@@@@@@@@@@ 2
16384  |                                     0
32768  |                                     0
65536  |                                     0
131072 |                                     0
262144 |                                     0
524288 |                                     0
1048576 |                                     0
2097152 |                                     0
4194304 | @@@@@@@@@@@@@@@@@@ 2
8388608 | @@@@@@@@@@@@@@@@@@ 2
16777216 |                                     0

#

```

On an idle system, if `txn_checkpoint` had no work to do, it might not produce any time distribution

graphs.

On a busy production system, the output will be longer. You might want to use the `dtrace -o` switch to specify an output file, or redirect the output, or run it in a script(1) session.

You might want to run this for a few minutes during a few relatively idle times and during the times of the day when the `txn_checkpoint` warnings are reported more often.

Live Example

Be careful when reading the time distribution graphs. Note how many times each call took each order of magnitude of amount of time. For example:

```
$ ./txnckpt.d -p 1140
2009 Apr 27 16:54:54 txn_checkpoint took 67509558730 nanoseconds
2009 Apr 27 16:58:10 txn_checkpoint took 134517395095 nanoseconds
^C
txn_checkpoint ran 2 times, duration (ms) min:67509 max:134517 avg:101013

lwp_mutex_timedlock called      5 times, total  5041 ms
pollsys called                  5 times, total  4134 ms
gtime called                    6 times, total    0 ms
lwp_mutex_wakeup called         16 times, total    0 ms
llseek called                   32 times, total    0 ms
write called                    32 times, total    3 ms
fdsync called                   48 times, total 176213 ms
pwrite64 called                 2139 times, total 11921 ms

Total time in syscalls          197315 ms
```

Two runs of `txn_checkpoint` took 202 seconds. The start times are not one minute apart as they should be. `fdsync` was called much less often than `pwrite64`, but it accounted for the vast majority of the time.

```
Time distributions for syscalls during txn_checkpoint (in nanoseconds):

gtime
value  ----- Distribution ----- count
  512  | 0
 1024  | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 5
 2048  | @@@@@@@ 1
 4096  | 0

llseek
value  ----- Distribution ----- count
 1024  | 0
 2048  | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 29
 4096  | @@@@ 3
 8192  | 0

lwp_mutex_wakeup
value  ----- Distribution ----- count
 1024  | 0
 2048  | @@@ 1
```

```

4096 | @@@@@@@@ 3
8192 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 11
16384 | @@@ 1
32768 | 0

write
value ----- Distribution ----- count
8192 | 0
16384 | @@@@@@@@@@ 7
32768 | 0
65536 | @@@@@@@@@@@@@@@@@@ 11
131072 | @@@@@@@@@@@@@@@@@@ 13
262144 | @ 1
524288 | 0

pollsys
value ----- Distribution ----- count
268435456 | 0
536870912 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 5
1073741824 | 0

lwp_mutex_timedlock
value ----- Distribution ----- count
65536 | 0
131072 | @@@@@@@@@@ 1
262144 | 0
524288 | @@@@@@@@@@ 1
1048576 | 0
2097152 | 0
4194304 | 0
8388608 | @@@@@@@@@@ 1
16777216 | 0
33554432 | 0
67108864 | 0
134217728 | 0
268435456 | 0
536870912 | @@@@@@@@@@ 1
1073741824 | 0
2147483648 | @@@@@@@@@@ 1
4294967296 | 0

pwrite64
value ----- Distribution ----- count
1024 | 0
2048 | @@@@@@@@@@ 352
4096 | @@@@@@@@@@ 412
8192 | @@@@@@@@@@ 305
16384 | @@@@@@@@@@@@@@ 544
32768 | @@@@@@@@@@ 303
65536 | @@@ 176
131072 | 24
262144 | 1
524288 | 0
1048576 | 0
2097152 | 0
4194304 | 1
8388608 | 2
16777216 | 0
33554432 | 1

```

67108864		1
134217728		4
268435456		6
536870912		6
1073741824		0
2147483648		1
4294967296		0

fdsync

value	----- Distribution -----	count
67108864		0
134217728	@@	2
268435456	@@@@@@	7
536870912	@@@@@@@@	10
1073741824	@@@@@@@@	10
2147483648	@@@@@@@@	9
4294967296	@@@	5
8589934592	@@@	3


```
17179869184 |@@          2
34359738368 |           0
```

`fdsync` was only called 48 times but it usually took between 0.2 seconds and 8.5 seconds. Three times it took over 8.5 seconds and twice it took over 17 seconds.

The two runs of `txn_checkpoint` took 202 seconds. 97 percent of that time was in `syscalls`. 89 percent of that was in `fdsync`.

So the question is: why is `fdsync` taking so long?

You can probably improve this by moving the `mboxlist` directory to a separate file system, if you have not done so already. You would want to tune that file system for write performance rather than space. That is, to get the performance you need, you will probably need to dedicate more space to the volume that you really need for the amount of data.

Additionally, you should work with Sun storage and performance experts to determine why `fdsync` is taking so long.

To Monitor the Mailbox Database Cache Size

1. Use the `imcheck` command to measure the cache hit rate.

```
imcheck -s mpool > imcheck-s.out
```

2. Find the cache information section in the output file, for example:

```
2MB 513KB 604B Total cache size.
1 Number of caches.
1          Maximum number of caches
2MB 520KB Pool individual cache size.
```

Then there will be several blocks of output – a summary and one for each database file – look for these lines in each block:

```
0 Requested pages mapped into the process' address space.
55339 Requested pages found in the cache (99%).
```

In this case, the hit rate is 99 percent. This could be optimal or, more likely, it could be that the cache is too large. To test, lower the cache size until the hit rate moves to below 99 percent. When you hit 98 percent, you have optimized the DB cache size. Conversely, if see a hit rate of less than 95 percent, then you should increase the cache size with the `store.dbcachesize` option.

As your user base changes, the hit rate can also change. Periodically check and adjust this parameter as necessary.

Chapter 27. 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](#)
- [Theory of Operations](#)
- [Options for MeterMaid](#)
- [Limit Excessive IP Address Connections Using Metermaid - Example](#)

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
<code>metermaid.enable</code>	<code>local.metermaid.enable</code>	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.
<code>metermaid.logfile.*</code>	<code>logfile.metermaid.*</code>	These settings are the same as those used by imap, pop, and other services. By default MeterMaid writes its log file into <i>msg-svr-base</i> /data/log/metermaid.
<code>metermaid.listenaddr</code>	<code>metermaid.config.listenaddr</code>	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)
<code>metermaid.maxthreads</code>	<code>metermaid.config.maxthreads</code>	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
<code>metermaid.port</code>	<code>metermaid.config.port</code>	This is the port to which MeterMaid listens for connections and to which MeterMaid clients will connect. Default: 63837
<code>metermaid.secret</code>	<code>metermaid.config.secret</code>	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.
<code>metermaid.sslusessl</code>	<code>service.metermaid.sslusessl</code>	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 <code>connectfrequency</code> 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 <code>check_metermaid.so</code> attempts to connect to MeterMaid, but it fails for some reason, during the next N seconds as specified by <code>metermaid_client.connectfrequency</code> (or <code>metermaid.mtaclient.connectfrequency</code> in legacy configuration), no additional attempts will be attempted. It prevents <code>check_metermaid.so</code> 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 <code>connecttimeout</code> 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 <code>timeout</code> 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 <code>metermaid.listenaddr</code> 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
<code>metermaid.local_table:*.data_type</code>	<code>metermaid.table:*.data_type</code>	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 <code>metermaid.local_table:*.options</code> below). Default: string
<code>metermaid.local_table:*.max_entries</code>	<code>metermaid.table:*.max_entries</code>	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 <code>quota_time</code> . Default: 1000

metermaid.local_table: *.options	metermaid.table. *.options	throttle	<p>This option is a comma-separated list of keywords that defines behaviour or characteristics for the table. Valid keywords are:</p> <ul style="list-style-type: none"> • <code>nocase</code> - When working with the data, all comparisons are done using a case-insensitive comparison function. (This option is valid only for string data). • <code>penalize</code> - After <code>quota_time</code> seconds, throttle will normally reset the connection count to 0, but if the <code>penalize</code> option is enabled, throttle will decrement the connection count by <code>quota</code> (but not less than 0) so that additional connection attempts will penalize future <code>quota_time</code> periods. For example, if <code>quota</code> were 5 with a <code>quota_time</code> 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 <code>quota</code> 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	<p>When a connection is received, it is counted against <code>quota</code>. If the number of connections received in <code>quota_time</code> 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</p>
metermaid.local_table: *.quota_time	metermaid.table. *.quota_time	throttle	<p>This specifies the number of seconds during which connections will be counted against <code>quota</code>. After this many seconds, the number of connections counted against the incoming address will be reduced depending on the <code>type</code> of this table. Default: 60</p>

metermaid.local_table: *.storage	metermaid.table. *.storage	throttle simple greylisting	MeterMaid can use two different storage methods, <code>hash</code> and <code>splay</code> . The default hash table method is recommended, but under some circumstances a splay tree may provide faster lookups. Default: <code>hash</code>
metermaid.local_table: *.table_type	metermaid.table. *.type	NA	MeterMaid supports three table types: <ul style="list-style-type: none"> • <code>throttle</code> (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 <code>quota_time</code> seconds. • <code>simple</code> - This type of table may be used to store arbitrary data referenced by a key. • <code>greylisting</code> - 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: <code>pt5m</code> (5 minutes)
metermaid.local_table: *.resubmit_time	metermaid.table. *.resubmit_time	greylisting	Specifies the ISO 8601 duration during which, but after <code>block_time</code> , 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: <code>pt4h</code> (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 <code>inactivity_time</code> from the last allowed delivery. Default: <code>p7d</code> (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: <i>table</i> .server_nickname	metermaid.mtaclient.remote_table. <i>table</i> .server_nickname	Specifies the nickname for a particular MeterMaid server that is responsible for the referenced <i>table</i> . The nickname is a short keyword, consisting only of letters, numbers, and underscores, that will be used in the <i>remote_server</i> option names. No default. Value must be supplied.
metermaid_client.remote_server: <i>nickname</i> .max_conns	metermaid.mtaclient.remote_server. <i>nickname</i> .max_conns	Specifies the maximum number of concurrent connections to the MeterMaid server referenced by <i>nickname</i> . Default: 3
metermaid_client.remote_server: <i>nickname</i> .server_host	metermaid.mtaclient.remote_server. <i>nickname</i> .server_host	Specifies the host name of the MeterMaid server referenced by <i>nickname</i> . Defaults to the value of <i>metermaid_client.server_host</i> .
metermaid_client.remote_server: <i>nickname</i> .server_port	metermaid.mtaclient.remote_server. <i>nickname</i> .server_port	Specifies the port number of the MeterMaid server referenced by <i>nickname</i> . Default: 63837
metermaid_client.remote_server: <i>nickname</i> .sslusessl	metermaid.mtaclient.remote_server. <i>nickname</i> .sslusessl	Specifies whether or not to use SSL for the connection to the MeterMaid server referenced by <i>nickname</i> . Defaults to the value of <i>metermaid_client.sslusessl</i> .

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
*|*|*|*|*
$:A$[/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 28. Implementing Greylisting by Using MeterMaid

Implementing Greylisting by Using MeterMaid

Topics:

- [What is Greylisting and How Does It Work?](#)
- [Basic Greylisting Implementation](#)
 - [Using Messaging Server 7 Update 4](#)
 - [Using Messaging Server Versions 6.3 Through 7 Update 3](#)
- [Enhancing Greylisting Functionality](#)
 - [Preloading the Greylisting Table with Outbound Transactions](#)
 - [Matching a Range of IP Addresses](#)
 - [Simplifying the Sender Address](#)
 - [Providing an Opt-In Mechanism](#)
 - [Whitelisting Based on User's Addressbook](#)
 - [Combining Functionality: A Complex Example](#)
- [Mapping Table Notes](#)

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|*|1|* \
$C$[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|*|1|*
$C$|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](#)
- [Matching a Range of IP Addresses](#)
- [Simplifying the Sender Address](#)
- [Providing an Opt-In Mechanism](#)
- [Whitelisting Based on User's Addressbook](#)
- [Combining Functionality: A Complex Example](#)

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|*|1|* \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylis,$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|*|1|* \
$C$[IMTA_LIB:check_metermaid.so,greylisting,greylis,$0|$|$X-CORRESPONDENT;$
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|1|local-recipient-address|recipient-mailUserGreyListOptIn|
TCP|$@*$@$*|$@$*|SMTP$@$*|MAIL|tcp_local|$_*|1|$_*|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|1|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|$_*|1|$_*|* \
$$[pabldap:///3?piEmail1?sub?(|(piEmail1=$1)(piEmail2=$1)(piEmail3=$1))[$
Now, for all other senders, do the normal greylisting check.
!
TCP|$@*$@*|*$@*|SMTP$@*|MAIL|tcp_local|$_*|1|$_*|* \
$$[IMTA_LIB:check_metermaid.so,greylisting,greylis,$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-channel|
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|1|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|@$*|@$*|*|@$*|SMTP$@$*|MAIL|tcp_local|$_*|1|$_*|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,greylis,$0.$1.$2|$4|$5]$CFIRSTATTEMPT
!
! (5)
!
  GREYLIST|$H*:$H*:$H*:$H*:$@H*:$@H*:$@H*:$@H*|*|*|* \
  $[IMTA_LIB:check_metermaid.so,\
  greylisting,greylis,$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 29. Using Predefined Channels in Unified Configuration

Using Predefined Channels in Unified Configuration

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 Unified Configuration](#), you should do so before reading this information. See [Configuring Rewrite Rules in Unified Configuration](#) for information about configuring the rewrite rules.

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](#)

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.
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 <code>/var/mail</code> . (Messaging Server does not support <code>/var/mail</code> access. User must use UNIX tools to access mail from the <code>/var/mail</code> 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 <code>imsimta</code> program interface.
reprocessprocess	These channels are used for deferred, offline message processing. The <code>reprocess</code> channel is normally invisible as a source or destination channel. The <code>process</code> 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 <code>tcp_smtp_client</code> , and runs as needed under the control of the Job Controller. <code>tcp_local</code> receives inbound messages from remote SMTP hosts. Depending on whether you use a <code>smarthost/firewall</code> configuration, either sends outbound messages directly to remote SMTP hosts or sends outbound messages to the <code>smarthost/firewall</code> system. Sometimes <code>tcp_local</code> gets mail from remote SMTP hosts via proxy or firewall. <code>tcp_local</code> is also sometimes used for internal relay activities. <code>tcp_intranet</code> receives and sends messages within the intranet. <code>tcp_auth</code> is used as a switch channel for <code>tcp_local</code> ; authenticated users switch to the <code>tcp_auth</code> channel to avoid relay-blocking restrictions. <code>tcp_submit</code> accepts message submissions, usually from user agents, on the reserved submission port 587 (see RFC 2476). <code>tcp_tas</code> 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 `sysexit.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 <i>command</i> 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 `imsimta qm` command 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. 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. 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 `msconfig edit conversions` command.

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 an `msconfig edit conversions` entry:

Conversions 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' "
```

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 `msconfig edit conversions` 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 entry in `msconfig edit conversions`. 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 qualifying parameters in [Conversions 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 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 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 `conversions` is modified, you must recompile the configuration if running a compiled 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 `conversions` 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` entry and incoming header show how to pass MIME information to the site-supplied program using environment variables.

msconfig edit conversions 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 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 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	<code>CHARSET_CONVERSION</code> - added as <code>;TAG=</code> field before <code>;CONVERT</code> .
1	2	<code>CONVERSION</code> - added as <code>;TAG=</code> field before <code>;CONVERT</code>
2	4	<code>FORWARD</code> - added just before current address (<code> </code> delim)
3	8	<code>ORIG_SEND_ACCESS</code> - added at end of probe (<code> </code> delim)
4	16	<code>SEND_ACCESS</code> - added at end of probe (<code> </code> delim)
5	32	<code>ORIG_MAIL_ACCESS</code> - added at end of probe (<code> </code> delim)
6	64	<code>MAIL_ACCESS</code> - added at end of probe (<code> </code> 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 <i>msg-svr-base/include/deprecated/pmdf_err.h</i>

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 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 `OVERWRITE-OPTION-FILE=1` in the appropriate conversions 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 `OVERWRITE-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 `OVERWRITE-OPTION-FILE=1` in the appropriate conversions 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 `OVERWRITE-OPTION-FILE=1` in the appropriate conversions 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
-----------	-------------

<i>Part 1: Qualifying Parameters (Specifies the parameters for which the message must match before it will be converted.)</i>	
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- <i>n</i>	Specifies the name of the Input MIME Content-Type parameter that must match for conversion; The <i>n</i> = 0, 1, 2.... is used to optionally pair the specified parameter name requirement with a value required by using IN-PARAMETER-VALUE- <i>n</i> with the same value of <i>n</i> .
IN-PARAMETER-VALUE- <i>n</i>	Specifies the value required of the input MIME Content-Type parameter whose name is specified in the corresponding IN-PARAMETER-NAME- <i>n</i> . 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- <i>n</i> and its value matches the value of this parameter. Wildcards allowed.
IN-PARAMETER-DEFAULT- <i>n</i>	Default value to use if the input MIME Content-Type parameter specified by the corresponding IN-PARAMETER-NAME- <i>n</i> is not present.
IN-DISPOSITION	Input MIME Content-Disposition to match for conversion.
IN-DPARAMETER-NAME- <i>n</i>	Specifies the name of the Input MIME Content-Disposition parameter that must match for conversion; The <i>n</i> = 0, 1, 2.... is used to optionally pair the specified parameter name requirement with a value required by using IN-DPARAMETER-VALUE- <i>n</i> with the same value of <i>n</i> .
IN-DPARAMETER-VALUE- <i>n</i>	Specifies the value required of the input MIME Content-Disposition parameter whose name is specified in the corresponding IN-DPARAMETER-NAME- <i>n</i> . 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- <i>n</i> and its value matches the value of this parameter. Wildcards allowed.
IN-DPARAMETER-DEFAULT- <i>n</i>	Default value to use if the input MIME Content-Disposition parameter specified by the corresponding IN-DPARAMETER-NAME- <i>n</i> 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 <code>CONVERSION_TAG</code> parameter.
<i>Part 2: Output Parameters (Specify the body part's post-conversion output settings.)</i>	
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- <i>n</i>	Specifies the name of a <code>content-type</code> parameter which will be set on the output body part.
OUT-PARAMETER-VALUE- <i>n</i>	Output MIME <code>Content-Type</code> parameter value corresponding to <code>OUT-PARAMETER-NAME-<i>n</i></code> .
PARAMETER-COPY- <i>n</i>	Specifies the name of a <code>content-type</code> parameter which should be copied from the input body part to the output body part.
OUT-DISPOSITION	Output MIME <code>Content-Disposition</code> if it is different than the input MIME <code>Content-Disposition</code> .
OUT-DPARAMETER-NAME- <i>n</i>	Output MIME <code>Content-Disposition</code> parameter name; <i>n</i> =0, 1, 2...
OUT-DPARAMETER-VALUE- <i>n</i>	Output MIME <code>Content-Disposition</code> parameter value corresponding to <code>OUT-DPARAMETER-NAME-<i>n</i></code> .
DPARAMETER-COPY- <i>n</i>	A list of the <code>Content-Disposition:</code> parameters to copy from the input body part's <code>Content-Disposition:</code> parameter list to the output body part's <code>Content-Disposition:</code> parameter list; <i>n</i> = 0, 1, 2,... Takes as argument the name of the MIME parameter to copy, as matched by an <code>IN-PARAMETER-NAME-<i>n</i></code> clause. Wildcards may be used in the argument. In particular, an argument of * means to copy all the original <code>Content-Disposition:</code> parameters.
OUT-DESCRIPTION	Output MIME <code>Content-Description</code> if it is different than the input MIME <code>Content-Description</code> .
OUT-MODE	Mode in which to read and store the converted file. This should be <code>BLOCK</code> (binaries and executables) or <code>TEXT</code> .
OUT-ENCODING	Encoding to apply to the converted file when the message is reassembled.
<i>Part 3: Action Parameters (Specify an action to take on a message part.)</i>	
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: <code>command="D:/tmp/mybat.bat"</code>
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"
<i>Part 4: Information Passing Parameters (Used to pass information to and from the site-supplied program.)</i>	
DPARAMETER-SYMBOL- <i>n</i>	Environment variable into which the Content-disposition parameter value, if present, will be stored; <i>n</i> = 0, 1, 2,... Each DPARAMETER-SYMBOL- <i>n</i> is extracted from the Content-Disposition: parameter list in order (<i>n</i> =0 is first parameter, <i>n</i> =2 second, etc.) and placed in the specified environment variable prior to executing the site-supplied program.
PARAMETER-SYMBOL- <i>n</i>	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; <i>n</i> = 0, 1, 2... Each PARAMETER-SYMBOL- <i>n</i> is extracted from the Content-Type: parameter list in order (<i>n</i> =0 is first parameter, <i>n</i> =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- <i>n</i> 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.

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 using `msconfig edit mappings` 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 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 entry by running `msconfig edit conversions` 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. If you are running a compiled configuration, 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` channel. 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 may 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 <i>out-channel</i> .
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

Stopped here

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 conversion entries. If a conversions entries exist, then the MTA will look in them 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 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 conversions 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 conversions 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 conversions entries. If conversions entries exist, then the MTA will look in them for an entry specifying a `SERVICE-COMMAND`, and if it finds such an entry, execute it. The conversions entries should be created using `msconfig edit conversions` and 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 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 30. Using Role-Based Access Control in Messaging Server Unified Configuration

Using Role-Based Access Control in Oracle Communications Messaging Server Unified Configuration

This information describes role-based access control and the required setup for Oracle Solaris OS where privileges are available.

Topics:

- [Overview of Role-Based Access Control](#)
- [Theory of Operations](#)
- [Setting Up and Using RBAC](#)
- [New Behavior](#)
- [Reference Information](#)

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 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. `pfexec` is used by the Messaging Server `start-msg`, `stop-msg`, and `imsimta` (through the `imtacli` program) commands, and the `job_controller`, to take advantage of role-based access controls.

For more information about role-based access controls, see `rbac(5)`.

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` directory, 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/MessagingServer.html` to the `/usr/lib/help/profiles/locale/C` directory.

This file is referenced by the Messaging Server profile definition. For example:

```
# 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.

```
# usermod -P 'Oracle Communications Messaging Server' mailsrv
```

6. 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. Set the `rbac` option to 1 to fully enable RBAC usage.

For example:

```
# ./msconfig set rbac 1
# ./msconfig show rbac
role.base.rbac = 1
```

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 31. 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](#)
- [Message Server iMIP Configuration Overview](#)
- [Configuring the iSchedule Channel for iMIP Messages in Unified Configuration](#)
- [Configuring the iSchedule Channel in Legacy Configuration](#)
- [Troubleshooting the iSchedule Configuration](#)

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](#)
- [Manually Configuring the iSchedule Channel in Unified Configuration](#)
- [Verifying the Calendar Server Configuration](#)
- [Modifying iSchedule Channel Options](#)

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 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 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 handling 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 32. Vacation Automatic Message Reply in Unified Configuration

Vacation Automatic Message Reply in Unified Configuration

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.

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](#)
- [Configuring Autoreply](#)
- [Vacation Autoreply Theory of Operation](#)
- [Vacation Autoreply Attributes](#)
- [Other Auto Reply Tasks and Issues](#)

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.

By default, the MTA evaluates vacation on the back-end store systems. However, because 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. One delivery address is generated for each valid `mailDeliveryOption` that is set in the recipient's LDAP entry. The patterns are defined by the setting of the `delivery_options` MTA option, which consists of a comma-separated list of *option=pattern* pairs. The default autoreply *option=pattern* pair is:

```
^*!autoreply=$M+$D@bitbucket
```

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
!	Enables 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.
*	Option is only evaluated for users, not groups.
@	Extracts preferred language information from various message header fields as well as from LDAP entries associated with envelope <code>From:</code> 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 <code>autoreply</code> delivery option. 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, add the # prefix to the autoreply pattern. This can be done by performing the following steps:

1. Use the `msconfig` command to determine the current setting of `delivery_options`:

```
msconfig
msconfig> show delivery_options
```

2. If the `delivery_options` MTA option is not set, determine the current default by using the `show -default` command and then cutting and pasting to set the option to that default:

```
msconfig> show -default delivery_options
delivery_options:
*mailbox=$M%$\%$2I$_+$2S@ims-ms-daemon,&members=*,*native=$M@native-daemon
set delivery_options
"*mailbox=$M%$\%$2I$_+$2S@ims-ms-daemon,&members=*,*native=$M@native-daemo
```

3. Once the option is set, you can use the `edit option` command to edit the option value and add the # if it is missing:

```
msconfig# edit option delivery_options
*mailbox=$M%$$2I$_+$2S@ims-ms-daemon,&members=*,*native=$M@native-daemon,
[Incomplete last line] 1 line, 254 characters
```

4. After changing the `delivery_options` option, use the `msconfig diff` command to verify that you have made the correct change. Then write the change out after you have made sure that it is correct:

```
msconfig# diff
< role.mta.delivery_options =
*mailbox=$M%$$2I$_+$2S@ims-ms-daemon,&members=*, *native=$M@native-daemon,
write -remark="Enable autoreply processing on relays"
```

Autoreply processing now takes place on the relays.

To Share Autoreply Information Between Relays

If the MTA performs 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 `mta.vacation_template` option. The `mta.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/sun/comms/messaging64/data/vacation/$3I/$1U/$2U/$
```

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` user ID.
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. Use the `msconfig` command to configure the `mta.vacation_template` option:

```
msconfig set vacation_template
file:///opt/sun/comms/messaging64/data/vacfiles/$3I/$1U/$2U/$U.vac
```



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 checked next.
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 generates 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 `mta.vacation_template` option. This option should be set to a `file: URL`.

Maintenance of these files is automatic and controlled by the `mta.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 is 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 `mta.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 `mta.ldap_personal_name`

Alias processing keeps track of personal name information specified in this attribute and uses 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 `mta.lda_start_date` MTA option to a different attribute name.

This attribute is read and checked by the code that generated the Sieve script. Vacation processing is 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 `mta.ldap_end_date` MTA option to a different attribute name.

This attribute is and checked by the code that generated the Sieve script. Vacation processing is 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 appear in the Sieve script as nonstandard `:echo` and `:reply` arguments to the vacation action. `echo` produces a "processed" message disposition notification (MDN) that contains the original message as returned content. `reply` produces a pure reply containing only the reply text. An illegal value does not manifest as any argument to the vacation action and this produces an MDN containing only the headers of the original message. 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 `mta.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 `mta.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 `mta.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 `mta.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 does not appear on a user entry, a default time-out is obtained from the `mta.autoreply_timeout_default` MTA option. The MTA can be instructed to use a different attribute for this information by setting the `mta.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 `mta.ldap_autoreply_subject` (normally `mailAutoReplySubject`), `mta.ldap_autoreply_text` (normally `mailAutoReplyText`), `mta.ldap_autoreply_text_int` (normally `mailAutoReplyTextInternal`), `mta.ldap_spare_4`, `mta.ldap_spare_5`, `mta.ldap_prefix_text` and `mta.ldap_suffix_text`.

It is expected that each attribute value has a different language tag value. If different values have the same tag value the choice between them is essentially random.

Other Auto Reply Tasks and Issues

This section describes auto reply tasks and issues not described in the configuration section.

Topic in this section:

- [To Send Autoreply Messages for Email That Have Been Automatically Forwarded from Another Mail Server](#)

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 has 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 does 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.

Because the recipient's address known to the remote system doing automatic forwarding is not 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 are 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 `mta.ldap_autoreply_addresses`.

Chapter 33. 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](#)
- [Hints and Tips](#)

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 than 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,$1,zen.spamhaus.org,Your$
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)
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,$1,zen.spamhaus.org.,\
host$ ($1)$ 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,$1,zen.spamhaus.org.,\
host$ ($1)$ 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,$1,zen.spamhaus.org.,\
host$ ($1)$ 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 than 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 34. 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
- Preventing Outbound Spam: Proactive Methods
- Preventing Outbound Spam: Reactive Measures
 - Blocking Submissions of Local Senders Who Might Be Spammers
 - Rate Limiting All Outgoing Email
 - Rate Limiting Only Outgoing Spam
 - Can Sieve Scripts Use Functions Defined Via Mapping Tables?
 - Reject/Discard All Outbound Spam
- Recovering From Phishing Attacks That Have Compromised User Accounts
- Greylisting Webmail
 - Requirements for Greylisting Webmail
 - Installing and Configuring Greylisting for Webmail
 - Troubleshooting Your Greylisting Deployment

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 receive 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 includes 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` directory.
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 following 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 35. 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](#)
- [Isolating Traffic From Messenger Express and Communications Express](#)
- [Creating a MeterMaid Configuration](#)
- [Creating Mappings Rules](#)
- [Testing and Debugging the Configuration](#)
- [Enable Rate-limiting at the Communications Express Level](#)
- [Restricting Access to New Source Channel](#)
- [Advanced Techniques](#)

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|
*|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 than 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 36. 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](#)
- [Secure Installation and Configuration](#)
- [Security Features](#)
- [Security Considerations for Developers](#)

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](#)
- [Installing Infrastructure Components](#)
- [Installing Messaging Server Components](#)
- [Post Installation Configuration](#)

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 Configuration 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 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:

- [Preventing Mail Relay](#)
- [Configuring SMTP Relay Blocking](#)

Starting with Messaging Server 7 Update 5, you can accomplish SMTP 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 [Certificate Based Authentication for Messaging Server](#) 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`, `plaintextmncipher`)

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](#)
- [How to Configure Messaging Server to Authenticate on Outbound SMTP Connections](#)
- [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 37. 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 about Sun's recommendation for how to deploy the Messaging Server MTA and anti-spam/anti-virus scanning systems:
http://www.sun.com/bigadmin/sundocs/articles/preferred_deploy_mta.jsp
- Call out to MeterMaid from the `FROM_ACCESS` mapping table passing as data the user authentication, rather than (or perhaps in addition 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 38. 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](#)
- [Planning Disk Capacity and Creating Volumes](#)
- [Configuring Messaging Server to Work with NetApp Filers](#)

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 39. 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](#)
- [Outbound IPv6 Connections](#)
- [Inbound IPv6 Connections](#)
- [TCP wrappers and MMP's Bad Guys Lists](#)
- [Connection Counters](#)
- [Mapping Tables](#)
- [Caveats](#)

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
<code>local.ipv6.in</code> (0 or 1)	<p>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.</p> <p>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.</p>
<code>local.ipv6.out</code> (0 or 1)	<p>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 <code>mshttpd</code>, 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.</p>
<code>local.ipv6.sortorder</code> (DEFAULT, A-AAAA, AAAA-A, A, AAAA)	<p>When <code>local.ipv6.out</code> 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 <code>getaddrinfo()</code> implementation. On Oracle Solaris, <code>getaddrinfo()</code> performs the re-ordering algorithm described in RFC 3484. See <code>getaddrinfo(3SOCKET)</code> for details.</p> <p>If you wish Messaging Server to attempt all A records before any AAAA records, then set this option to have the value <code>A-AAAA</code>. To attempt all AAAA records first, use <code>AAAA-A</code>. To only honor AAAA records and thus only make IPv6 outbound connections, use the value <code>AAAA</code>. To only honor A records and thus only make IPv4 outbound connections, use <code>A</code>. To attempt the records in the order returned by <code>getaddrinfo()</code>, use the value <code>DEFAULT</code>.</p> <p>The default setting for this option is <code>DEFAULT</code>.</p>

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 40. 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](#)
- [VCS Installation and Configuration Notes](#)
- [Unconfiguring High Availability](#)

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:

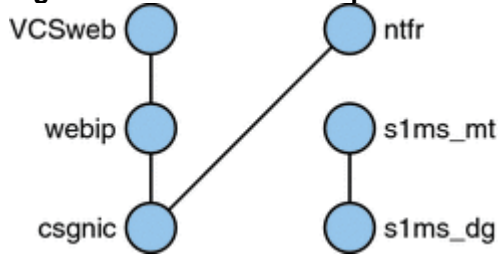
```
/opt/VRTSvcs/bin/hagui
```

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.

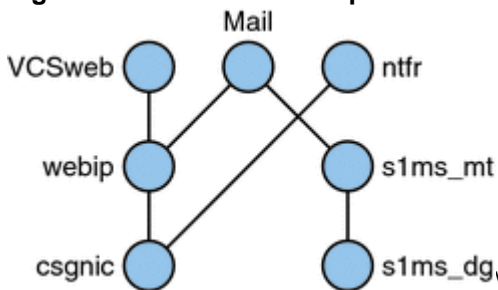
11. As VCS HA agent is part of Messaging Server installation, run `config-vcsha` to copy agent files to VCS configuration.


```
<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.
ConflInterval	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 return OFFLINE 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.