SWAAG-19.1

M-Switch Advanced Administration Guide

Table of Contents

1 Software version

This guide is published in support of Isode M-Switch R19.1. It may also be pertinent to later releases. Please consult the release notes for further details.

2 Readership

This guide is intended for administrators who plan to configure and manage advanced features of the Isode M-Switch message switch. For detailed information on Operating M-Switch, use the complementary volume the M-Switch Operator's Guide . Administrators who plan to configure and manage more basic features of the Isode M-Switch message switch use the complementary volume the M-Switch Administration Guide .

3 How to use this guide

You are advised to read through Chapter 1, *Overview*, before you start to set up your messaging system.

4 Typographical conventions

The text of this manual uses different typefaces to identify different types of objects, such as file names and input to the system. The typeface conventions are shown in the table below.

Object	Example
File and directory names	*isoentities*
Program and macro names	mkpasswd
Input to the system	`cd newdir`
Cross references	see Section 5, “File system place holders”
Additional information to note, or a warning that the system could be damaged by certain actions.	Notes are additional information; cautions are warnings.

Arrows are used to indicate options from the menu system that should be selected in sequence.

For example, **File** → **New** means to select the **File** menu and then select the **New** option from it.

5 File system place holders

Where directory names are given in the text, they are often place holders for the names of actual directories where particular files are stored. The actual directory names used depend on how the software is built and installed. All of these directories can be changed by configuration.

Certain configuration files are searched for first in *(ETCDIR)* and then *(SHAREDIR)*, so local copies can override shared information.

The actual directories vary, depending on whether the platform is Windows or UNIX.

Name	Place holder for the directory used to store...	Windows (default)	UNIX
(ETCDIR)	System-specific configuration files.	*C:\Isode\etc*	*/etc/isode*
(SHAREDIR)	Configuration files that may be shared between systems.	*C:\Program Files\Isode\share*	*/opt/isode/share*
(BINDIR)	Programs run by users.	*C:\Program Files\Isode\bin*	*/opt/isode/bin*
(SBINDIR)	Programs run by the system administrators.	*C:\Program Files\Isode\bin*	*/opt/isode/sbin*
(EXECDIR)	Programs run by other programs; for example, M-Switch channel programs.	*C:\Program Files\Isode\bin*	*/opt/isode/libexec*
(LIBDIR)	Libraries.	*C:\Program Files\Isode\bin*	*/opt/isode/lib*
(DATADIR)	Storing local data.	*C:\Isode*	*/var/isode*
(LOGDIR)	Log files.	*C:\Isode\log*	*/var/isode/log*
(CONFPDUSPOOLDIR)	Large PDUs on disk.	*C:\Isode\tmp*	*/var/isode/tmp*
(QUEDIR)	The M-Switch queue.	*C:\Isode\switch*	*/var/isode/switch*
(DSADIR)	The Directory Server's configuration.	*C:\Isode\d3-db*	*/var/isode/d3-db*

6 Support queries and bug reporting

A number of email addresses are available for contacting Isode. Please use the address relevant to the content of your message.

- For all account-related inquiries and issues: customer-service@isode.com. If customers are unsure of which list to use then they should send to this list. The list is monitored daily, and all messages will be responded to.
- For all licensing related issues: license@isode.com.
- For all technical inquiries and problem reports, including documentation issues from customers with support contracts: support@isode.com. Customers should include relevant contact details in initial calls to speed processing. Messages which are continuations of an existing call should include the call ID in the subject line. Customers without support contracts should not use this address.

- For all sales inquiries and similar communication: sales@isode.com.

Bug reports on software releases are welcomed. These may be sent by any means, but electronic mail to the support address listed above is preferred. Please send proposed fixes with the reports if possible. Any reports will be acknowledged, but further action is not guaranteed. Any changes resulting from bug reports may be included in future releases.

Isode sends release announcements and other information to the Isode News email list, which can be subscribed to from the address: http://www.isode.com/company/news-signup.php [http://www.isode.com/company/contact.php]

7 Export controls

Many Isode products use TLS (Transport Layer Security) to encrypt data in transit. This means that these products are subject to UK Export Controls.

For some countries (at the time of shipping this release, these comprise all EU countries, United States of America, Canada, Australia, New Zealand, Switzerland, Norway, Japan), these Export Controls can be handled by administrative process as part of evaluation or purchase. For other countries, a special Export License is required. This can be applied for only in context of a purchase order for those Isode products.

You must ensure that you comply with these Export Controls where applicable, i.e. if you are licensing or re-selling Isode products.

The TLS feature of Isode products is enabled by a TLS Product Activation feature. This feature may be turned off, and Isode products without this TLS feature are not export controlled. This can be helpful to support evaluation of Isode products in countries that need a special export license.

Isode products are used to administer sensitive data and so Isode strongly recommends that all operational deployments of Isode products use the export-controlled TLS feature.

All Isode Software is subject to a license agreement and your attention is also called to the export terms of your Isode license.

Chapter 1 Overview

This chapter covers the advanced features of the Isode M-Switch, giving detailed descriptions of the advanced features and components.

1.1 What is the Isode M-Switch?

M-Switch is a high-performance, versatile Message Transfer Agent (MTA), which can be installed on either Windows or UNIX platforms. It is the main component in a messaging system and supports:

- Internet messaging
- X.400 messaging
- A mixture of the two variants, converting messages from one form to the other.

The MTA consists of:

- The Queue Manager (`qmgr`)
- Channels
- Protocol listeners for messaging entering the MTA (`iaed` or `smtpsrvr`)
- M-Vault, used to hold configuration
- Management tools (GUIs and command line)

Chapter 2 Channel Overview

The processing of messages by M-Switch is carried out by channels. This chapter describes the different kinds of channels, and how they are configured.

2.1 Overview

This chapter is principally concerned with configuring channels which perform most of the work of the core MTA. Channels are normally configured using MConsole to add, delete or modify configuration values in the Directory. When the Queue Manager starts up it reads the directory entries (having obtained the necessary Directory connection information from its *mtaboot.xml* file) and creates an *mtatailor.tai* file which contains the runtime MTA configuration information and is used by all other M-Switch programs to get the MTA configuration values. If you are using MConsole to set up MTA tailoring, the information in this chapter explains in greater detail how the various values are used. If you are not using MConsole, this chapter gives the detail you will need to construct the *mtatailor.tai* file using a standard editor.

2.2 Channel Types

The following types of channel can be configured in M-Switch:

Table 2.1. Channel Types

Value	Type of channel
in	An incoming channel
out	An outgoing channel
both	Both incoming and outgoing
check	A channel which performs message checking
corrector	A channel which acts as a webserver to allow clients to perform message corrections
housekeeper	A channel which performs general MTA housekeeping
shaper	A channel which performs message conversion

2.3 How channels work

Channel processes can be:

- started by the queue manager (using the '`program`' value configured for the channel)
- part of a standalone daemon or service, e.g. `acp142`
- accessed through a process using the X.400 MT API or XAPI XMT API

The channel name is set in the command line when started by the queue manager, and is part of the configuration of the process for the other types.

The channel process opens a connection to the queue manager which is used for communication between the two. The channel process:

- requests messages to process
- informs the queue manager of the results of operations
- can send to the queue manager details of messages during transfer

The queue manager:

- requests the channel to connect to a peer MTA (for connection oriented channels)
- gives the channel a message plus a set of recipients to process
- relays commands from a console for controlling the channel

The results of an operation can cause the queue manager to perform other steps. There are three kinds of temporary problem:

- a channel failure results in no processing on the channel for an interval
- an MTA failure results in the MTA being made inactive for an interval
- a message failure causes the message to be delayed for an interval

These delays are visible in MConsole. If a message is being processed when a channel or MTA failure occurs, the message can also be delayed.

The channel can tell the queue manager that a report is needed for the message. The queue manager will then send the message to the housekeeper channel.

Housekeeper Channel

The housekeeper channel performs a number of operations on messages on behalf of the queue manager:

- Generation of Reports/DSNs on error or successful delivery
- Conversion of Reports to DSNs
- Redirecting a recipient to another address
- Causing the routing of a recipient to be re-evaluated
- Non-delivering a message at the request of an operator
- Discarding a message at the request of an operator
- Generate a warning DSN for a message Generate IPNs/MDNs for a boundary MTA

2.4 Channel Pairing

Many of the channels within the MTA naturally fall into pairs. For instance, it is usual to have a channel handling inbound messages (responder or server), and a corresponding channel handling outbound messages (initiator or client).

As this is a common occurrence, one channel definition in the *mtatailor.tai* file can be used to define both channels. Such a channel should be marked as type both . Where both sides of the channel require a parameter, such as a table, there is an explicit in and out value.

Channel pairing can bring several benefits. If the Queue Manager knows about paired channels, it can make some optimizations. When a message is received for an MTA on the inbound side of a paired channel, it can, if there are messages waiting to go out on that channel, assume that this MTA has just come up and schedule a retry immediately.

For X.400 channels, an alternative to channel pairing is to configure a channel for two-way alternate mode of operation, which enables a responder or initiator to both send and receive messages over a single association. This is described in the Section entitled ***X.400 P1 Channel***.

A side effect of channel pairing can help with authorization. An incoming channel usually has a fixed name. However, if a number of channels are defined in the *mtatailor.tai* file with a key of that value, then all of these channels are potentially usable. In this case, the MTA that the message was received in is looked up in each of the associated mtatable table. The first channel to match on both key and the contents of the mtatable is chosen. Authorization is then done on this basis. One of the channels may not have an mtatable associated with it, in which case this is the default channel to use.

NOTE: This gives the first match, rather than the best possible match of tables.

The format of the mtatable is identical to the domain table. The value on the right hand side is only used in determining matches. It must however be a legal value.

As an example, consider a host on the Internet which wishes to authorize usage on the basis of whether the user is local or remote. This might be achieved by the following:

```
chan smtp-local key=“smtp”,mtatable=localhosts,
...
chan smtp-internet key=“smtp”
```

The incoming channel then claims to be smtp. If the MTA the message is received from is present in the localhosts table, then smtp-local is used as the channel, and authorization done on that basis. Otherwise it is assumed to be smtp-internet and potentially different authorization is applied.

On a smaller site, this feature can be used to map many channels onto one.

Chapter 3 Routing

The routing of messages by M-Switch is the process by which a message is either delivered, or relayed to another MTA which is nearer to system which can deliver the message.

3.1 Lookup policies

The way M-Switch performs routing can work in several different ways. At the highest level there are two basic models:

- Directory-based routing
- Table-based routing

In Directory based routing, routing information is held in the Directory and is read by the MTA components (such as channels) as required when processing messages etc. This is the model which is appropriate for most deployments of M-Switch and is generally recommended by Isode. Although information is held in the Directory, tables are still used in a Directory-based routing configuration. These tables can be held in the Directory or in files.

In Table-based routing all routing information is held in files on disk. Some M-Switch features are not available when using Table-based routing. Table-based routing can be appropriate for some very simple configurations, but Isode generally do not recommend the use of Table-based routing.

LASER routing (named after the Internet working group that developed a specification with this name that did not get published as a standard) is a subset of Directory-based routing which uses LDAP search operations to find routing and/or delivery information for Internet addresses. It is possible for this information to be held in a separate LDAP directory from the rest of M-Switch's routing and configuration information.

M-Switch tailoring configuration can also be held in the Directory or maintained exclusively in configuration files. Although the *mtatailor.tai* file is always held on disk, most configurations will hold the MTA tailoring information in the Directory with the Queue Manager downloading the information from the Directory to create the *mtatailor.tai* file for other M-Switch programs to use.

The various options for routing models are controlled using the Lookup Policy configuration of the MTA. (Although you can also configure different Lookup Policies on a per channel basis).

Possible Lookup Polices are:

Table 3.1. Lookup Policies

Value	Effect
table [=prefix]	Use tables for lookup
ds	Use X.500 for lookup
dns [=prefix]	Use the DNS for lookup
dns-tbl [=prefix]	Use the DNS for lookup with routing override in tables
queue	Queue the message
nssoft	Continue with the lookup policies even if the previous nameserver lookup fails

dns-ds	Use the DNS for initial lookup. If necessary, continue address resolution using the Directory
dns-laser	Use DNS for routing and LASER lookup in the Directory for delivery information
table-laser	Use tables for routing and LASER lookup for delivery
dns-table-laser	Use DNS for initial lookup. If necessary continue using tables. Use LASER for delivery information

If the optional *prefix* is specified in any of the policies, it will be prepended to the standard tables, *domain, or, channel, aliases* and *users*.

For example, if the policy is *table=eg* then the tables that may referenced when evaluating that policy are *eg-domain, eg-or, eg-channel, eg-aliases* and *eg-users*. The default value for *default_lookup_policy* is table.

Note: these Lookup Policies can be combined into a sequence if required. The routing process will then try each policy in the sequence in turn until the address in question can be successfully routed.

3.2 Wildcard routing

The general routing algorithm described in M-Switch Administration Guide allows the administrator to configure a portion of the address space to be routed in a particular way (e.g. towards an MTA). In the example, this model of routing works well because the OR Address prefix (e.g. `/O=AnotherCompany/PRMD=Isode/ADMD= /C=GB/`) can be expressed as a Routing Tree node.

However, there are occasions when this works less well, and a large number of **Routing Tree** nodes need to be configured to represent a set of similar OR addresses. In these cases it is possible to specify a **Filter** in a **Routing Tree** node, which is used to route subordinate OR Address attributes.

Wildcard Routing is only supported in Directory-based configurations: the filters are held as attributes in entries within a Routing Tree.

Routing Filters should be configured using the Routing Tree entry editor within MConsole's Switch Configuration View.

The following sections describe the two sorts of Filters available:

- Routing Filters
- Redirect Filters.

3.2.1 Routing Filters

Routing filters allow a set of addresses to be routed using the information in an arbitrary routing node. Routing filters are only used to cause routing to relay to a non-local MTA – a Routing filter is not used to route for local delivery. (See Section 3.2.2, “Redirect filters” if you wish to do this.)

Table 3.2. Routing filter example

Address	Destination Location
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGDZPZX/	BRISTOL

/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGWZPZX/	LUTON
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGKKZPZX/	GATWICK
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPFZPZX/	GLASGOW
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPHZPZX/	EDINBURGH

Then the following routes would be needed:

Table 3.3. Routing filters required

/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGD*/	London MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGW*/	London MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGKK*/	London MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPFZPZX/	Scotland MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPHZPZX/	Scotland MTA

A Routing filter consists of:

- an OR Address attribute type which is an OID (e.g. `mHS-OrganizationalUnitName`)
- weight (used to prioritise multiple matching Filters)
- regular expression (used to match OR Address attribute value)
- DN of a RoutingTree node

In the above example, the attribute types are all `mHS-OrganizationalUnitName` have the default weight of 5; have regular expression values which are either a simple string or include regular expression (regex) features; and have a Routing Tree node (London or Scotland).

The weights are not important in this example, because the regexes are configured so that no more than one regex can match. In configurations where more than one value can match, the weight determines which Filter is used (only the first matching filter will be used).

The regexes are of two types:

- simple string
- simple character and special regex chars such as
- “^” start of line
- “$” end of line
- “.” any character

If the regex is absent, all values match.

The Routing Tree node allows the routing information in any Routing Tree node to be used to route address which match the filter. In the above example, addresses which do not match any of the Filters are routed to a fallback MTA.

Routing filter example:

```
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGDZPZX/ <--- address in BRISTOL
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGWZPZX/ <---address in LUTON
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGKKZPZX/ <---address in GATWICK
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPFZPZX/ <--- address GLASGOW
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPHZPZX/ <--- address in  EDINBURGH
```

Then the following routes would be needed (using regex syntax):

```
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGD.*/ ----> London MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGGW.*/ ----> London MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGKK.*/ ----> London MTA

/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPFZPZX/ ----> Scotland MTA
/C=XX/A=ICAO/P=EG/O=AFTN/OU=EGPHZPZX/ ----> Scotland MTA
```

To do this you would add the **Routing Filters** as shown in Figure 3.2, “Example of a Routing Filter.”

Note how in this example External X.400 MTAs for London and Scotland have been added. Routing Tree nodes for /C=XX/A=ICAO/P=EG/O=AFTN/OU=London (which has MTA Information pointing at the London External MTA) and /C=XX/A=ICAO/P=EG/O=AFTN/OU=Scotland/ ((which has MTA Information pointing at the London External MTA) have also been added.

Figure 3.1. Example of MTA Info.

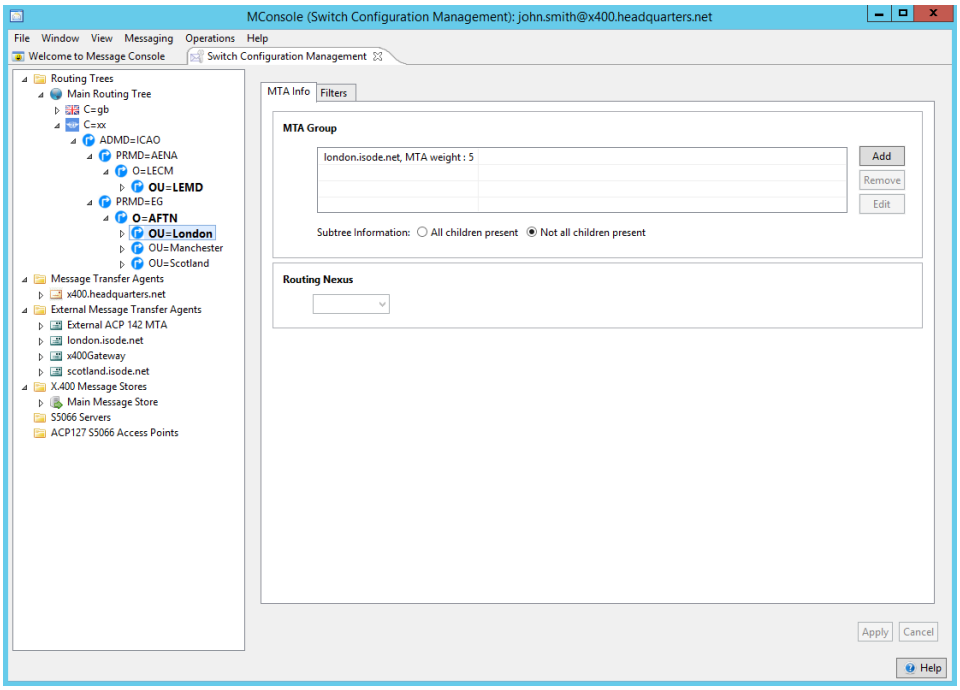
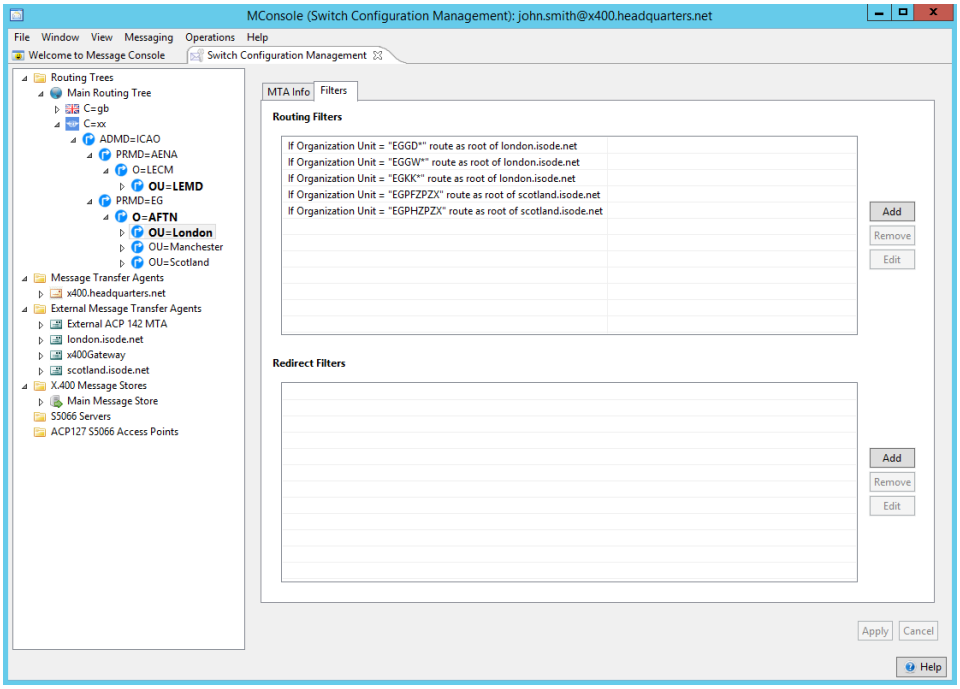


Figure 3.2. Example of a Routing Filter.



A routing filter consists of

- an OR Address attribute type which is an OID (e.g. **mHS-OrganizationalUnitName**)
- weight (used to prioritise multiple matching filters)
- regular expression (used to match OR Address attribute values)
- DN of a **Routing Tree** node

In the above example, the attribute types are all **mHS-OrganizationalUnitName**; have the default weight of 5; have regular expression values which are either a simple string or include regular expression (regex) features; and have a **Routing Tree** node (London or Scotland).

The weights are not important in this example, because the regexes are configured so that no more than one regex can match. In configurations where more than one value can match, the weight determines which filter is used (only the first matching filter will be used).

The regexes are of two types

- simple string
- simple character and special regex chars such as
  - ^ start of line;
  - $ end of line
  - . any character

If the regex is absent, all values match.

All OR Address attribute values are treated as a regex. So special regex chars must be quoted. So if you wanted, for instance, to match

```
/OU=Fleet(Rear)/
```

you would have to configure the value:

```
/OU=Fleet\(Rear\)/
```

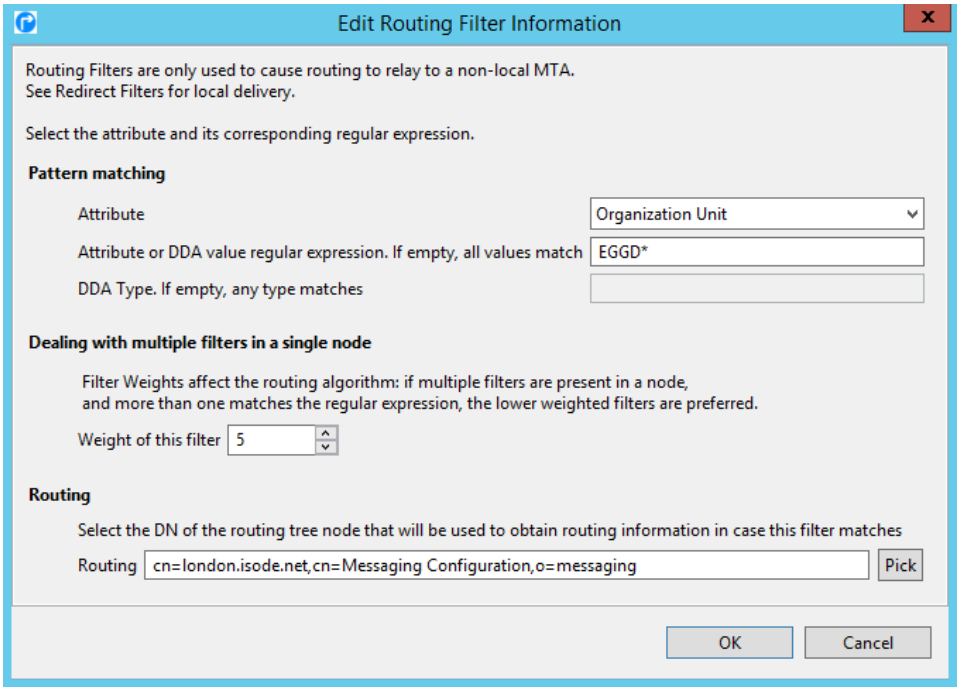
> **Note:** PCRE regex's are used which are rather more feature-full than described in this section

The **Routing Tree** node allows the routing information in any **Routing Tree** node to be used to route address which match the filter.

In the above example, addresses which do not match any of the **Routing Filters** are routed to a fallback MTA.

When creating or modifying a **Routing Filter**, the following window is displayed.

Figure 3.3. Creating or modifying a filter.



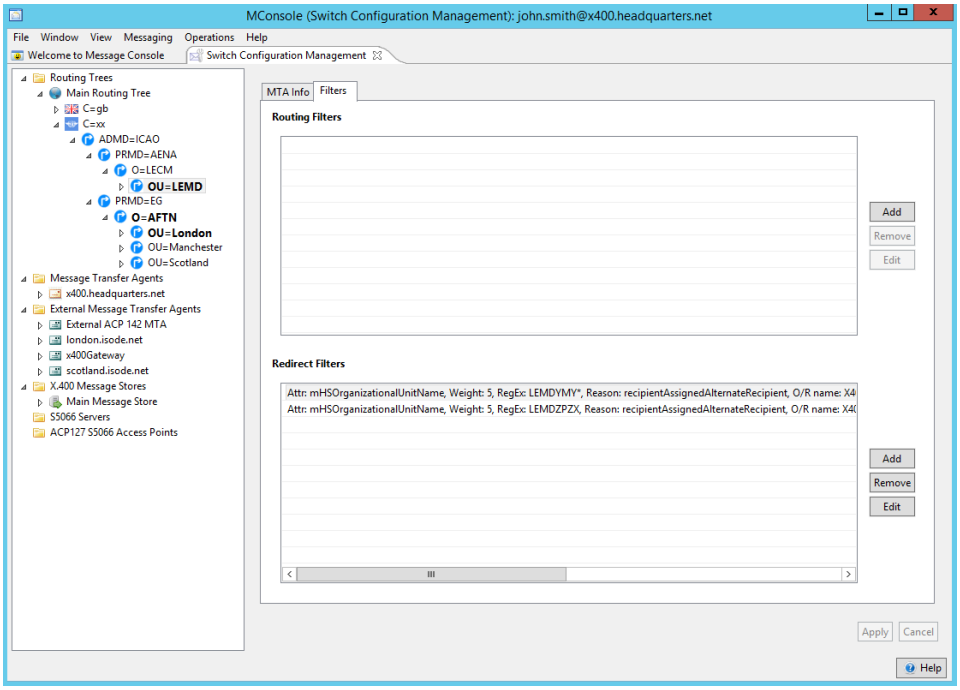
3.2.2 Redirect filters

When you wish to use wildcard routing, you can also use **Redirect Filters**. These are the same as **Routing Filters**, except that instead of using the Routing Information in an arbitrary **Routing Tree** node, the address is redirected to a new ORName. In all other respects **Redirect Filters** and **Routing Filters** work in the same way.

In this example,

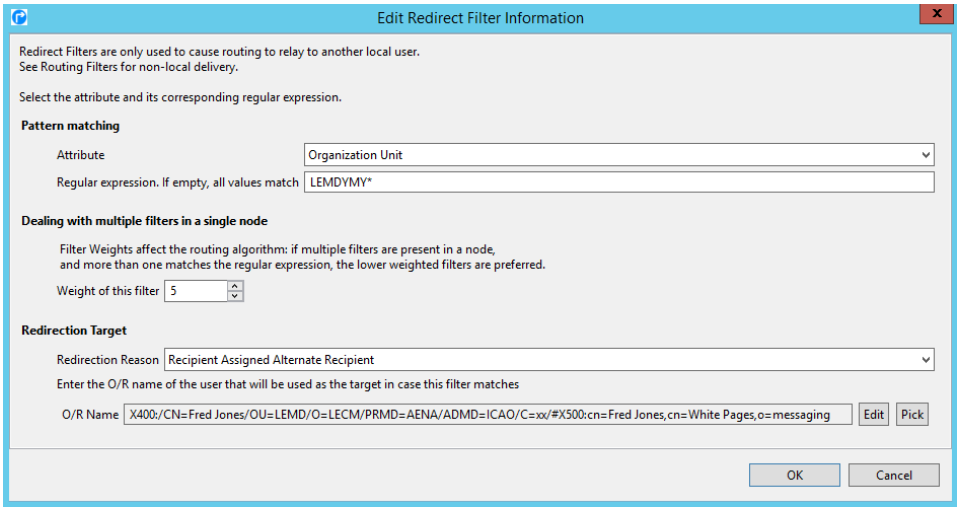
```
/C=XX/A=ICAO/P=AENA/O=LECM/OU=LEMD/CN=LEMDYMY*/ ---> is redirected
to /CN=Fred Jones/
/C=XX/A=ICAO/P=AENA/O=LECM/OU=LEMD/CN=LEMDZPZX/ ---> is redirected
to /CN=John Smith/
```

Figure 3.4. Redirect Filter.



The figure below shows how to edit a redirect filter to cause the Redirect Filter to operate as needed in this example.

Figure 3.5. Redirect Filter.



The above configuration results in the redirect required as shown using ckadr below.

```
C:\Program Files\Isode\bin>ckadr -x
    "/C=XX/A=ICAO/P=AENA/O=LECM/OU=LEMD/CN=LEMDYMYA/"
/C=XX/A=ICAO/P=AENA/O=LECM/OU=LEMD/CN=LEMDYMYA/ ->  (x400)
    /CN=Fred Jones/OU=LEMD/O=LECM/PRMD=AENA/ADMD=ICAO/C=XX/

/C=XX/A=ICAO/P=AENA/O=LECM/OU=LEMD/CN=LEMDYMYA/ -> (rfc822)
    "/CN=Fred Jones/OU=L EMD/O=LECM/PRMD=AENA/ADMD=ICAO/C=XX/"
    @x400.headquarters.net

Delivered to x400.headquarters.net by p3deliver (weight: 0)
```

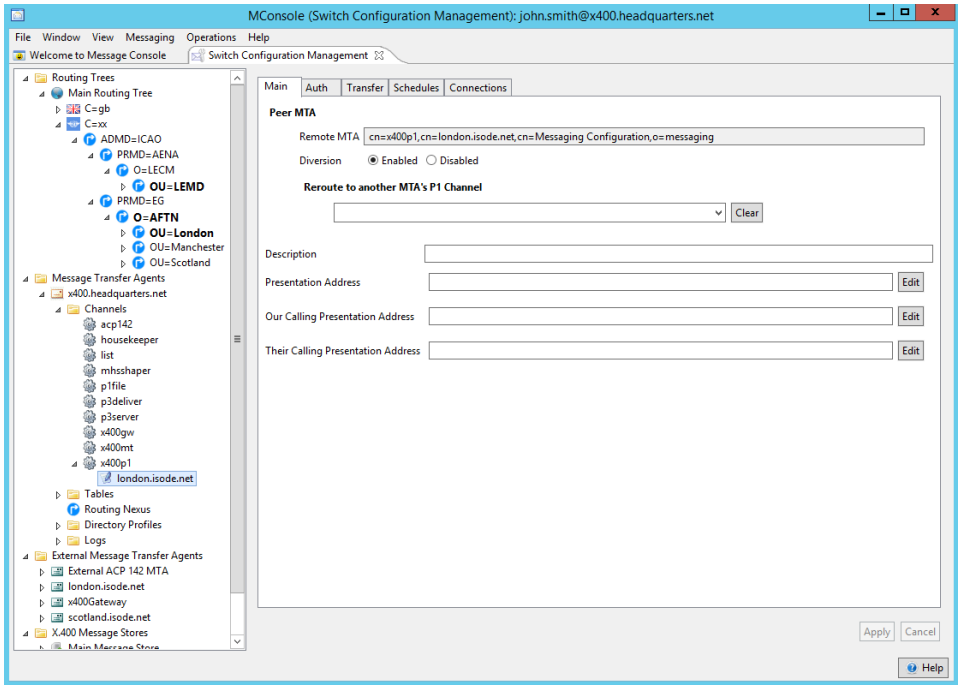
3.2.3 Rerouting

There are other ways of configuring routing behaviour which override the simple model outlined above.

3.2.3.1 Diversions

When it is known that an MTA is temporarily unavailable, it can be convenient to label this in the configuration and allow a fallback MTA to be used without attempting the failed MTA. To do this, select the **Peer Connection** of the MTA which is unavailable and select the **Diverted** checkbox.

Figure 3.6. Setting an MTA as unavailable.



3.3 Redirect Filters

Redirect Filters are only supported in Directory-based configurations: the filters are held as attributes in entries within a Routing Tree. Redirect Filters should be configured using the Routing Tree entry editor within MConsole's Switch Configuration View.

When you wish to use wildcard routing, use Redirect Filters. These are the same as Routing Filters, except that instead of using the Routing Information in an arbitrary Routing Tree node, the address is redirected to a new ORName. In all other respects Redirect Filters and Routing Filters work in the same way.

3.4 Multiple Institutions on One MTA

The MTA contains several features that enable a site to behave as multiple mail hubs and/or as a central mail gateway between the external world and the internal subdomains. The features in question are:

- the local key in the `domain` and/or `or` tables
- the external qualifier in the `aliases` tables
- the `internal` and `external` parameters to the MIME header normalisation filter

NOTE: In general, the local key should be used symmetrically in the `domain` and `or` tables (based on the `or2rfc` and `rfc2or` mappings). If they are not symmetric, unexpected results may be obtained.

To explain how these features may be used, this section introduces three possible configurations. Each one slightly more complex than its predecessor and each one is built from its predecessor. These examples consider the situation of two domains, the main site name, `widget.co.uk`, and a subdomain under that site, `admin.widget.co.uk`.

3.4.1 The MTA Acting As Two Different, Independent Mail Hubs

This is the situation where one system is acting as two distinct domains. The main domain `widget.co.uk` has the information relating to its local users in the tables named `users` and `aliases`, and the subdomain `admin.widget.co.uk` has the information relating to its local users in the tables `admin-users` and `admin-aliases`.

There is no intersection between the `widget.co.uk` name space and the `admin.widget.co.uk` name space so these tables can be independently maintained.

The MTA has to be informed that `admin.widget.co.uk` is a local domain. This is done via the relevant entry in the domain table, for example:

```
admin.widget.co.uk:local=admin
```

and/or the or table:

```
OU$admin.O$widget.PRMD$widget.ADMD$ .C$GB:local admin
```

3.4.2 The MTA and Multiple Namespaces

This is the situation where the MTA is acting as one complete namespace composed of two separate realms, and is hiding multiple namespaces behind a single externally-visible namespace; for example, all users have mail addresses of the form:

```
user@widget.co.uk
```

This section outlines an example where the RFC-822 domain to be used externally is `widget.co.uk`, and there is more than one internal domain: `admin.widget.co.uk`, `sales.widget.co.uk`, and so on. These are separate namespaces, so that it is possible to have:

```
fred@admin.widget.co.uk
fred@sales.widget.co.uk
```

and have these addresses refer to separate people; in this case they must be translated to different external addresses.

NOTE: It is this aspect that makes the solution complicated - if your internal domains are essentially there for routing, and there are no name conflicts, then the local-parts (to the left of the '@') can be treated as coming from a single namespace.

Each namespace needs a name, although one can be used as the default; it is normal for the external domain to be the default namespace. For convenience the internal namespaces will be called after the domains, that is, `admin` for `admin.widget.co.uk` and sales for `sales.widget.co.uk`.

Then the system would be tailored with:

```
loc_dom_site widget.co.uk
```

In the domain table, you would have:

```
widget.co.uk: local
admin.widget.co.uk: local=admin
sales.widget.co.uk: local=sales
```

and so on.

The local keyword means that the local part of the address is significant, and to be used for routing. The value identifies the namespace, that is, the tables that get used when looking up the local-part. There will be the default `aliases` and `users` tables, used for the default namespace; in addition, for each named namespace, you need:

- <namespace>-aliases
- <namespace>-users
- <namespace>-channel

The <namespace>-channel table has the same format as the normal `channel` table, and should contain as keys those MTAs that appear in the <namespace>-users table. It could be a link to the normal `channel` table.

Consider two users:

Table 3.4. local user table

Internal	External
smithj@admin.widget.co.uk	J.A.Smith@widget.co.uk
smithj@sales.widget.co.uk	J.B.Smith@widget.co.uk

These users would need entries as outlined below:

The main `aliases` table:

```
J.A.Smith: alias smithj@admin.widget.co.uk 822
J.B.Smith: alias smithj@sales.widget.co.uk 822
```

The function of these two entries is to direct mail for the external mail address to the appropriate internal address.

No entries are needed in the main `users` table.

The `admin-aliases` table:

```
    smithj: synonym J.A.Smith@widget.co.uk 822 external
```

The `sales-aliases` table:

```
    smithj: synonym J.B.Smith@widget.co.uk 822 external
```

The function of these entries is to translate the internal address to the external address. The external keyword is important here. It stops the looping that would otherwise occur between the entries in the <intdom>*-aliases tables and the main aliases table. It also illustrates why `admin` and `sales` need to be in separate namespaces: there are entries with the same key mapping to different information.

The `admin-users` table:

```
    smithj: smtp intmta1.co.uk
```

The `sales-users` table:

```
    smithj: smtp intmta2.co.uk
```

These entries are only illustrative: the users table gives the channel/MTA for delivery for that user. It is assumed that these users are not actually local to the MTA, therefore the MTA in the user's entry will not be `loc_dom_mta`. (In this case the channel is not relevant, but it is a useful mnemonic to set the channel to that channel which will be used for the MTA). The internal domains may be reached by a single gateway, in which case that will be the MTA name used. There needs to be a suitable entry in `admin-channel` for `intmta1.co.uk` and one in `sales-channel` for `intmta2.co.uk` (or whatever actual values for the MTA key are used).

The result of these entries will be as follows:

- Mail submitted by `smithj@admin.widget.co.uk` will have the originator (as in the `SMTP MAIL FROM:<>` command, or - suitably translated - in X.400 P1 envelope) changed to `@A.Smith@widget.co.uk@`.
- Mail which arrives at the MTA addressed to `J.A.Smith@widget.co.uk` will be sent to `intmta1.co.uk` addressed (in the envelope) to `smithj@admin.widget.co.uk`.

Note that to change header fields, you must carry out the changes to header normalisation as described in Internet Message Filters.

Chapter 4 Table Based Configuration

This section covers in detail the way MTAs, Channels and Tables can be configured using tables.

4.1 MTAs

An M-Switch MTA configuration is accessed by M-Switch components using the *mtatailor.tai* file. This is usually created by the Queue Manager on startup from information held in the Directory. Table-based configuration without the use of the Directory is supported but is not appropriate for most deployments.

Variables can be one of three types:

- standard *mtatailor* variables (e.g. *"postmaster postmaster@example.com"*)
- Isode variables (e.g. *"fsync TRUE"*)
- pp variables (e.g. *"set snmp=true"*)

This section describes the tailoring variables which can appear in Mconsole and the *mtatailor.tai* file when generated by the Queue Manager. Each of the variables is described with the following information:

- The name of variable as known in the *mtatailor.tai* file.
- A reference to the Mconsole tab under which the variable appears.
- A description of the variable, the M-Switch feature it controls and its syntax.
- Whether the variable is mandatory or optional.
- Whether the variable is
  - An ordinary tailor variable.
  - An "Isode" variable (overriding any value in the *isotailor* file).
  - A "pp" variable.
  - A variable which can only be configured in the Directory.

There are a large number of tailoring variables that can be referenced in the *mtatailor.tai file*. These tailoring variables are divided into two types, mandatory variables and optional variables. They are listed and described in the sections below. For convenience, they are organized by the Mconsole tab in which they appear.

4.1.1 Main

These variables appear in the Main tab of Mconsole.

Full MTA Domain (Mandatory; Tailor(`loc_dom_mta`)): This is the full domain name of the local MTA. It is used for trapping routing loops and so should be globally unique. For example: `ourmta.example.com`

Note that this value is not configured in the directory. If the configuration is downloaded from the directory by the queue manager, the value of this is set from the local host's name.

Description (Optional; Directory): A textual description of the MTA.

Postmaster (Mandatory; Tailor(`postmaster`)): This is the RFC 822 address of the local mail system administrator. Typically of the form: `Postmaster <postmaster@example.com>`

Default Domain (Internet) (Mandatory; Tailor(`loc_dom_site`)): This is the full domain name of the local site. This is used to reference the site, and may refer to a group of MTAs collectively. For example: `example.com`

Default Domain (X.400 O/R) (Mandatory; Tailor(`loc_or`)): This is the local OR-address defaults given in X.400 RFC 1327 encoding form. It is used to fill in missing default components and for tracing fields. It is not recognised as the local X.400 domain by default, and must be marked in the or table.

For example: `"/OU=Sales/O=attlee/PRMD=TestPRMD/ADMD= /C=GB/"`

If the name contains spaces or other special characters, it must be quoted as above.

Enable SNMP sub-agent (Optional; PP(`set snmp`)): Selecting this option makes the Queue Manager connect to the master SNMP agent, enabling monitoring of M-Switch using SNMP. For example: `set snmp=true`

4.1.2 Delivery

These variables appear in the Delivery tab of MConsole. They configure the behaviour of the MTA when handling messages and determining whether the message should be delivered or non-delivered.

Some of the variables in this section are based on the priority of the message, The variables that can be set vary depending on whether standard X.400 or Military priorities are in use. This can be set using Mconsole, but this does not set a configurable value, merely changing the options presented as configurable.

The priority of a message is determined from values in the message envelope or from heading fields for an Internet message. In decreasing order of preference:

- Value from the message envelope
- Value from MMHS-Primary-Precedence heading field
- Value from MMHS-Copy-Precedence heading field
- Value from Priority heading field

Timeout From Submission (Optional; Tailor(`returntime`)): This is the time after which to expire an undelivered message, probe or report. The time is calculated from the time a message, probe or report is submitted to the MTS, or from the deferred time.

The times may be specified as *hours* , *hours : minutes* , *or hours : minutes : seconds*. For example, 15 minutes is specified as `0:15`.

Note: Messages that pass through the list channel may have their priority changed: Internet messages are always set to low priority; the behavior for X.400 messages depends on the Distribution List policy, which allows the existing priority to be preserved or a new priority level to be substituted.

The CCITT F.410 recommendations for these values are: high priority 2 hours; normal priority 6 hours; low priority 12 hours.

The Isode defaults for X.400 priorities are: high priority 36 hours; normal priority 3 days; low priority 6 days. The Isode defaults for military priorities are: override priority 6 hours; flash priority 36 hours; immediate priority 2 days; priority priority 3 days; routine 4 days; deferred 6 days. The default for reports is 36 hours and for probes is 1 day.

The order of the values in the configured value is (with the military priority in parentheses):

- Message Normal (Priority) : Isode default 72
- Message High (Flash) : Isode default 36
- Message Low (Deferred) : Isode default 144
- Report : Isode default 36 Probe : Isode default 24
- Message Override : Isode default 6
- Message Immediate : Isode default 48
- Message Routine : Isode default 96

These defaults would be configured in the *mtatailor.tai* file as:

```
returntime 72 36 144 36 24 6 48 96
```

If the MTA attempts to timeout a message which is currently active, because a transfer or delivery attempt is in progress, then the timeout is delayed for a short period. By default this is 60 seconds. This may be configured using the internal variable `timeout_retry_interval` ; for example: `set timeout_retry_interval=120`

Timeout From Arrival (Optional; Tailor(`returntimemin`)): This is similar to `returntime`, except that the time is calculated from the time a message arrives at the MTA, as opposed to the time a message is submitted to the MTS. The purpose of this variable is to permit the local delivery or transfer of messages which have exceeded the time given in `returntime`. This could occur if the message has taken a long time to reach the MTA, or if an old RFC 822 message has been resent.

The value takes the same form as `returntime`, that is hours, `hours:minutes`, or `hours:minutes:seconds`, and could be set to something like 35 minutes (0:35). As with `returntime`, specifying one value sets the time for normal priority, and default values for high and low priority messages are then automatically set based on the value given. These defaults can be overridden as described for the `returntime` variable. The initial default value is zero.

Setting `returntimemin` values greater than `returntime` values results in only the time from arrival at the MTA being used. As a guideline for X.410, the CCITT F.410 recommendations give target transfer times for 95% of messages as 0:35 for normal priority, 0:10 for high priority and 02:24 for low priority.

The Isode defaults are set to 10 minutes for all priorities which is configured as:

```
returntimemin "0:10" "0:10" "0:10" "0:10" "0:10" "0:10" "0:10" "0:10"
```

Detect Loops After (Traces) (Optional; Tailor(`maxhops`)): This is a number indicating the number of trace fields a message may contain. If it has more than this number, the message will be rejected as looping. The default is 25.

Detect Loops After (Loops) (Optional; Tailor(`maxloops`)): This is a number indicating the number of times a message may pass through this MTA before being rejected. The default is 5.

Warning Messages (Maximum) (Optional; Tailor(`nwarnings`)): This is the maximum number of warnings to send. The default is 2. When the value is greater than 0, the default value of the ESMTP parameter includes delay.

Warning Messages (Interval) (Optional; Tailor(`warninterval`)): This is the time in hours after which to send a warning to the sender of the message telling him or her that the message is stuck in the MTA. The default is 24 hours.

4.1.3 Routing

Items configured in the Routing tab of MConsole configure the way in which M-Switch components route messages using Directory-based Routing. The information configured here is held in the Directory and accessed directly by the M-Switch components.

No *mtatailor.tai* variables are set as a result of configuring values in the Routing tab.

4.1.4 Lookup

Items configured in the Lookup tab of MConsole configure the routing strategy of the MTA.

Lookup Policies (Mandatory; Tailor(`default_lookup_policy`)): This configures a sequence of Lookup policies which determine the way in which M-Switch components such as channels perform routing. The possible options and their effect is described in Lookup Policies.

An example of how this is configured in a MIXER configuration is:

```
default_lookup_policy ds table-laser dns
```

Most of the remaining values all configure Directory access. By definition there is no need for table-based ways to configure these values.

Directory Access (Access Method) (Optional; PP(`set x500_access`)): This configures the way in which Directory is accessed by M-Switch components. This be either DAP (the default) or LDAP. This could be configured by: `set x500_access=ldap`

Connection Details (Host) (Optional; PP(`set ldap_host`)): This configures the LDAP host when using LDAP for Directory access. For example: `set ldap_host=localhost`

Connection Details (Port) (Optional; PP(`set ldap_port`)) : This configures the LDAP port when using LDAP for Directory access. For example: `set ldap_port=19389`

Connection Details (Authentication Mechanism) (‘Optional; PP(`set ldap_sasl_mech`)): configures the SASL mechanism when using LDAP for Directory access. For example: `set ldap_sasl_mech=CRAM-MD5`

Connection Details (User) (Optional; PP(`set ldap_sasl_user`)): configures the SASL userid when using LDAP for Directory access with a SASL authentication mechanism. For example: `set ldap_sasl_user=myid`

Connection Details (Password) (Optional; PP(`set ldap_sasl_pass`)): configures the SASL password when using LDAP for Directory access with a SASL authentication mechanism. For example: `set ldap_sasl_pass=pw`

Subaddressing (Recipient delimiter) (Optional; PP(`set recipient_delimiter`)): This variable enables a subaddressing separator to be specified so that routing of the RFC822 address can be performed using using only the section of the local-part of the address preceding the delimiter. Commonly this delimiter is "+" and the technique is known as "plus-addressing". For example: `set recipient_delimiter=+`

The SASL Server Configuration variables are used when configuring the MTA to offer SASL authentication to SMTP clients via the SMTP AUTH command.

SASL Configuration Entry (Optional; PP(`set sasl_config_entry`)): This variable selects where in the Directory the MTA looks for SASL configuration information. The information read from the entry is then used by inbound SMTP channels when performing SASL authentication in response to the AUTH command. The SASL Configuration Entry itself may contain various attributes which control the way in which SASL is used (by the DSA as well as by the MTA), and can be edited using SODIUM if necessary.

Authentication Mechanism (Optional; PP(`set sasl_ldapdb_mech`)): This allows configuration of the SASL mechanism which the MTA will use when binding to the DSA to perform authentication of clients.

Password (Optional; PP(`set sasl_ldapdb_pw`)): The password to be used when binding to the DSA.

ID for SASL Bind (Optional; PP(`set sasl_ldapdb_id`)): The SASL ID to be used when binding to the DSA.

DN for Simple Bind (Optional; PP(`set sasl_ldapdb_dn`)): The DN to be used when binding to the DSA using simple (i.e. non-SASL) authentication.

4.1.5 Authorization

The Rules and Groups which can be configured on the Authorization tab of MConsole control various aspects of the Queue Manager's handling of incoming connections and messages.

Full descriptions of Rules, Groups and Rule Filters are given in Section 9.1, "M-Switch Authorization".

4.1.6 Queue Manager

Items configured in the Queue Manager tab of MConsole configure the way the Queue Manager controls many of the activities of the MTA for which it is responsible.

Values are in seconds unless otherwise specified.

Full descriptions of the QMGR Parameters are given in M-Switch Administration Guide.

4.1.7 Security

These values configure the way in which M-Switch uses supports authentication. This includes:

- SOM client connections to the Queue Manager
- SMTP client connections the SMTP Server

4.1.7.1 TLS Configuration

Enable TLS for SOM Protocol (Optional; PP(`set qmgr_tls`)): Set this to "1" to enable use of TLS for SOM connections.

Path for TLS identity files (Optional; PP(`set tls_path`)): Path for TLS identity files containing Digital Identities. This is used by both the Queue Manager and SMTP server processes.

Name of PEM file containing CA certificates (Optional; PP(`set tls_cafile`)): To set the location of the PEM file used to hold the list of trusted Certificate Authorities. This is used by both the Queue Manager and SMTP server processes.

4.1.7.2 SASL Configuration

Enable SASL for SOM protocol (Optional; PP(`set qmgr_sasl`)): Enable the use of SASL for SOM connections.

Disable SASL PLAIN and LOGIN: (Optional; PP(`set qmgr_sasl_noplain`)): This option specifies whether plaintext SASL mechanisms (PLAIN, LOGIN) are allowed for SOM connections.

Example of SASL and SOM configuration:

```
    set qmgr_tls=1
    set tls_path=/opt/tls-id
    set tls_cafile=file.pem
    set qmgr_sasl=1
    set qmgr_sasl_noplain=1
```

4.1.8 Advanced (including internal variables)

These are a miscellaneous set of variables which configure a range of M-Switch properties which are either not usually necessary to configure or do not fit into the other sets of variables.

Admin alt. recipient (Optional; Tailor(`administration_assigned_alternate_recipient`)): This variable allows an undeliverable local address to be redirected to the user given here, provided the originator has allowed this. The value should be in RFC 822 format. Possible uses of this feature might be to deliver all failed messages to the postmaster for advice, or to deliver to a special program that returns fuzzy matches. It is important that the users (or processes) to which undeliverable messages are redirected ensure that all such message originators are notified that the message has not been delivered to the intended recipient, as this redirection is used instead of the normal message report indicating delivery failure.

Archive (Optional; PP (`set archive_dir`)): The directory where the MTA writes archived messages. This can contain H and %M, which are replaced with the date, hour and minute of the time of the archiving. Archiving can be used in conjunction with the Message Audit Database so that message content can be displayed when using Message Tracking.

Example *mtatailor* entry:

```
set archive_dir=/var/isode/archive/%D
```

Channel address (Optional; PP(`set qmgr_chan_address`)): This configures the address, in `<host>:<port>` format, on which the Queue Manager listens for channels.

Example *mtatailor* entry:

```
set qmgr_chan_address="localhost:18001"
```

Configuration reload interval (Optional; PP(`set qmgr_config_time`)): This is the time interval use by the Queue Manager to check if the Directory configuration has been updated and that a new *mtatailor* file needs to be created. (The Queue Manager reads the mt_serial attribute to do this).

Discard probes (Optional; PP(`set discard_probes`)): This causes all probes (X.400 messages with no content) to be discarded and no report generated.

Example:

```
set discard_probes=true
```

DSA address (Optional; Tailor(`dsa_address`)): This configures the Presentation Address used to connect to the Directory.

Examples:

```
dsa_address TELEX+00728722+RFC-1006+03+127.0.0.1+19999
dsa_address /Internet=127.0.0.1+19999
```

dsaptailor (Optional; Tailor(`dsap_tailor`)): The configures a dsaptailor file to use to configure Directory access in place of the default.

Example

```
dsaptailor /etc/isode/dsaptailor
```

Fsync (Optional; Tailor(`fsync`)): This is a boolean variable which controls the use of the fsync (2) call for unlocking files which have been locked using the flock , fcntl or lockf lock styles. fsync is normally used when critical files are written before passing back a handshake over protocol. However, it can be switched off by a value of no if you consider that it is too expensive.

Example:

```
fsync FALSE
```

Hard disk space to leave free (Optional; Tailor(`diskuse`)): This parameter is used to limit the disk space consumed by the MTA, by setting the disk space that should remain free. It is measured in megabytes. This can be disabled by setting the value to zero.

Hard disk percentage to leave free (Optional; Tailor(`diskuse`)): This parameter is used to limit the disk space consumed by the MTA, by setting the percentage of the disk space that should be free. This can be disabled by setting the value to zero.

The two figures presented by Mconsole are combined into a single tailoring variable. The disk that is checked is the partition holding the queue (quedir).

Example:

```
diskuse 1024 95
```

Isode variables (Optional; Isode()): This allows arbitrary Isode tailoring variables (i.e. those found in the *isotailor* file) to be overridden.

Example:

```
isode threads 2
```

Log directory (Optional; Tailor(`logdir`)): This configures the directory in which M-Switch is to write log files. This overrides the default value and values in *isotailor*.

Example:

```
logdir /var/isode/newlog
```

No delete (Optional; PP(`set no_delete`)): This variable allows messages with which M-Switch has completed processing for all recipients to be preserved in the queue. This is usually only of value when attempting to solve a problem with a particular message. If unset, messages are deleted by whichever channel happens to set the status of the final recipient to "done".

Example:

```
set no_delete=true
```

Operation rate time (Optional; PP(`set qmgr_oprate_time`)): This sets the smoothing time for calculation of the operation rate - i.e. the period over which the operation rate is averaged.

Example:

```
set qmgr_oprate_time=1.1
```

Prevent return of contents (Optional; PP(`set prevent_roc`)): Setting this to "`true`" causes the P3 and x400p1 channel never to return content when generating non-delivery reports. I t also forces the X.400 value "`content-return-request`" to "`false`" in messages, and message being sent of SMTP will have `RET=HDRS`.

Example:

```
set prevent_roc=true
```

4.1.8.1 PP Internal variables

Values configured here allow arbitrary, extensible values to be configured internally to the MTA, of which Mconsole is unaware. Possible values, some of which are configured by Mconsole, are described elsewhere in this section.

The following is a list of possible values which may be used:

- **allow_duplicates** : By default duplicate recipients are removed. Setting this variable to any value allows duplicate recipients to be retained.
- **default_p1bind_password**: If the password is not present in the configured credentials to be used in a P1 bind, the default of " " (single space) is used. This allows a different value to be used. Specify " " (i.e. two double quotes and an empty string) to configure a zero length password. **Note:** M-Switch treats single space and a zero length password as matching when verifying P1 bind credentials.
- **internal_cache_size**: All M-switch programs which carry out Directory lookups when performing Directory-based routing have an internal cache of the result of the lookup. This variable allows the size of this cache to be configured. The default is 10000.
- **internal_cache_timeout**: All M-switch programs which carry out Directory lookups when performing Directory-based routing have an internal cache of the result of the lookup. This variable specifies the age limit in seconds of items read from this cache to be configured. The default is 300 (seconds).
- **local_deref_alias**: The Directory based routing depends on dereferencing aliases in the Directory. Some non-Isode Directories do not dereference the alias. If set to TRUE, this variable forces the M-Switch lookup code to attempt to dereference the alias itself.
- **mixer_space_replace** : The first character of the value is used to replace spaces when generating string forms of OR addresses for use on the Internet side of a MIXER gateway.
- **no_nicepn** : Do not use the encoding of personal name in a MIXER gateway which is defined in RFC 2156 4.1.2.
- **nocheckcontent** : When the MTA is configured to perform content checking (e.g. for viruses or spam), certain content types can be exempted so they are not checked. The value is a list of content type for example:

```
set nocheckcontent=p2,p22,oid.1.3.26.0.4406.0.4.1
```

- **PASSWD**: On Unix platforms, channels (such as 822-local) which need to read the passwd file to perform authentication by default use */etc/shadow*. The name of the file can be overridden by this variable.

- **pn_separator**: Change the separator used in the RFC 2156 4.1.2 encoding of personal names.
- **RESPATH** : Used by the 822-local channel on Windows.
- **USRPATH** : Used by the 822-local channel on Windows.

Queue depth (Optional; Tailor(`queuestruct`)): The level of sub-directories to create in the message queue directory.

Queue fan out (Optional; Tailor(`queuestruct`)): This specifies the fan out of the queue.

The two fields which MConsole allows you to set are combined into a single tailoring variable which is used to control the structure of the queue. This is only useful to change if you are expecting very large queues (>2000 messages in the queue at one time.) It takes two numbers. The first number specifies the fan out of the queue. With a value of 100 , messages will be put in 100 sub-directories of the main queue file. This cuts down the searching of the main directory but is only useful for very large queues. The second optional number is the level of sub-directories to create. By default this is one, indicating an indirection of one directory. Again this parameter should not be changed unless extremely large queues are expected (>50,000.). Example to configure a Queue Fan Out of 1, and a Queue Depth of 15:

```
queuestruct 1 15
```

Queue directory (Optional; Tailor(`queuedir`)): Override the default location of the M-Switch Queue Directory where the MTA keeps the messages it is holding on disk. It needs to be an absolute pathname. Example:

```
quedir /var/isode/switch
```

Redirect on routing failure (Optional; PP(`set redir_routing_failure`)): Redirect messages to admin-assigned-alternate recipient on Routing Failure. Example:

```
set redir_routing_failure=true
```

Set environment variable (Optional; Tailor(`setenv`)): This allows setting of environment variables. The syntax is setenv key = value . Example:

```
setenv MY_ENV_VAR=my_value
```

SOM address (Optional; PP(`set qmgr_som_address`)): The host and port on which the Queue Manager listens for requests from SOM clients. Example:

```
set qmgr_som_address="somhost.example.com:8888"
```

System mailfilter (Optional; Tailor(`mailfilter`)): Sets the default name of the system mailfilter file used by the `822-local` channel.

Table directory (Optional; Tailor(`tbldir`)): This configures where M-Switch searches for tables which are held in filestore (i.e. those tables which have a flags value of `dbm` or `linear`). Example:

```
tbldir /var/isode/table
```

Trash lifetime (Optional; PP(`set trash_lifetime`)): Non-message files older than this lifetime are deleted from the Queue. Expressed as `<integer><units>` where units are: `s:` seconds; `m:` minutes; `h:` hours; `d:` days. Example setting to 30 minutes:

```
set trash_lifetime=30m
```

Unset environment variable (Optional; Tailor(`unsetenv`)): This allows unsetting of environment variables. The syntax is unsetenv key = value . Example:

```
unsetenv MY_ENV_VAR=my_value
```

X.400 MTA (Optional: Tailor(`x400mta`)): The sets the MTA name used to represent this MTA in tracing fields. Example:

```
x400mta tracemtaname
```

4.1.8.2 Tailor Variables Not Configurable in Mconsole

default_lookup_timeout : The default time in seconds to spend on non-table lookups (i.e. Directory and DNS). The default is 30 seconds.

lockstyle: This is the style of locking to be used when it is necessary to lock a file. It may be one of the following:

Table 4.1. lockstyle

Value	Meaning
flock	Use the flock (2) system call
fcntl	Use the fcntl (2) system call
locknt	Windows file locking

The default style on Unix is flock but may be changed if you have doubts about the interaction of flock (2) and NFS . This is a run time configuration variable, and it will depend on what platform you are running as to whether all the above will be supported. The file locking mechanism will always be available, though.

timeout_retry_interval: This is described under `returntime` in the previous section, and is expressed in seconds.

4.1.9 Table Entries

This section describes how the tables of aliasing and addressing information, referenced by the channel programs, are defined in the *mtatailor* file.

Generally each table's entry is comprised of a line with several key/value pairs. The line starts with the keyword `tbl` which is followed by a string setting the default name for the table and the plain text file in `tbldir` . Then come the key/value pairs, described below.

name=<value>: Name this table with the given value (to override the default - see above). This is included only for completeness as it is set by first argument; it is not normally used. The name is used in all references to the table.

file=<value>: The tables contents are found in the given file (to override the default).

show=<value>: A descriptive string used when printing messages about this table, mainly for logging purposes. It defaults to the same as the table name.

flags=<value>: A set of flags that specifies how this table operates. It should be one of the following:

- `dbm`: This table is stored in the database for fast access (the default).
- `linear`: This table is searched with a linear pass through the file in `tbldir` (this is slow, but does not require rebuilding the database, and changes take immediate effect). Linearly searched files are still built into the database.
- `empty`: This flag which indicates there is no file associated with the table.

override=<value>: This allows values in the table to be overridden, or indeed if the table is small the whole table to be specified in the *mtatailor* file. Each occurrence of this keyword adds a new key/value pair to a list. The format should be exactly the same as in a file with a key, a colon character (:) and a value.

An example of table tailoring is shown below:

```
tbl aliases show="Aliases: mapping -> local id",
flags=dbm, file=local-aliases
tbl domain show="Mapping domain key -> full domain/MTA",
flags=dbm, override="widget.co.uk:local"
tbl or show="Mapping O/R Address -> MTA",flags=dbm,
tbl or2rfc show="RFC 987: X.400 -> RFC 822", flags=dbm
tbl rfc2or show=" RFC 987: RFC 822 -> X.400",flags=dbm
tbl channel show="Binding MTA -> Channels",flags=dbm
tbl auth.qmgr show="Queue Manager authorization",flags=linear
```

4.2 Channels

Channel Tailoring

Channels are perhaps the most complex aspect of tailoring. There are several types of channels. Input channels carry messages into the system. Output channels carry messages out of the system. Reformatting channels change the message structure in the queue. Checker channels scan messages for viruses and spam. Housekeeper channels generate Delivery Reports, DSNs and warning messages, and carry out other Switch maintenance tasks.

Each channel entry in the *mtatailor.tai* file consists of the keyword `chan` followed by a value which, by default, names the channel and the program associated with it. This is followed by a list of key/value pairs which provide more information on the channel.

A number of the values can be applied to a pair of channels differently depending on whether the channel is being used in outbound or inbound mode. Where the distinction is significant, there are separate tailor variables prefixed by `in` and `out`.

Channel configuration is normally performed using MConsole, and for that reason the descriptions of channel tailoring variables are presented in groups which match the various tabs on MConsole's Channel Properties pane, with the labels used by MConsole and the corresponding *mtatailor* key name where different from the MConsole label.

4.2.1 Main

Channel Name : the name of the channel. Corresponds to the value after the chan keyword in the *mtatailor.tai* file.

How it appears in the logging : a short phrase describing the channel. If this string starts with the keywords with or via then this value is included in Received lines generated for RFC 822 messages.

Description : longer description of channel. Stored in the Directory only and not used by the MTA.

Channel Type : corresponds to the type keyword. This parameter is mandatory and indicates the type of channel this is. This must be set to one of the values shown below:

Table 4.2. Channel Type

Value	Type of Channel
in	An incoming channel.
out	An outgoing channel.
both	Both incoming and outgoing.
check	A channel which performs message checking.
housekeeper	A channel which performs general MTA housekeeping.
shaper	A channel which performs message conversion.

MConsole further restricts the available set of choices for this field, depending on the nature of the channel being edited.

Access : The type of access this channel requires; the value is either `mta` , in which case the channel must be run by a trusted userID, or else `mts` in which case authentication of messages is enabled. The default is `mta`. To explain more fully the implications of the value set here:

- If the program that is run for the channel is setuid, and the channel is not `access=mts`, an error is reported.
- If `access=mts` and type=out or `type=both`, then the owner needs to be root (otherwise it needs to be the `ppuser`).
- If `access=mts` and `type=out` or `type=both`, and the program is not setuid, a warning is given. Basically if the program is setuid root, the channel needs to be `access=mts` and vice-versa.

Basically if the program is setuid root, the channel needs to be `access=mts` and vice-versa.

Outbound Protocols : This is a list of application contexts the channel can transfer, and corresponds to the `appcont` keyword. This field can only be set for a protocol channel. Where a protocol channel can only handle one application context, the value is simply displayed, and cannot be modified. This parameter configures the ways in which the channel can connect out to an external MTA

The values which appear in the *mtatailor.tai* file are the string representations of the Object Identifiers defining the application contexts, as follows:

Table 4.3. Outbound Protocols

Value	OID	Meaning
P1 (normal)	2.6.0.1.6	X.400 P1 1988 transfer
P1 (1988) X.410	1.3.6.1.4.1.453.5.5	X.400 P1 1988 X.410 mode transfer
P1 (1984)	1.3.6.1.4.1.453.5.1	X.400 P1 1984 mode transfer
P3	2.6.0.1.1	P3 delivery
SMTP	1.3.6.1.4.1.453.5.2	SMTP message transfer
MTS Gateway	1.3.6.1.4.1.453.5.8	Channel uses MTS Gateway API for message transfer

Value	OID	Meaning
P1 File	1.3.6.1.4.1.453.5.7	Channel uses P1 File convention for message transfer
ACP142	1.3.6.1.4.1.453.5.9	ACP142 message transfer

4.2.2 Programs

Program to run: The program associated with this channel, i.e. the actual binary to run (overrides the default value, which is the channel name). Corresponds to the `prog` keyword. May include command line arguments to be passed to the channel process.

Key : A list of keys (comma separated) by which this channel is known. This can be used to map several logical channels onto one. See Section 2.4, “Channel Pairing”. Note that a key is an equally valid way to refer to a channel. When referencing a channel, the first match on key or name is taken. If specifying several keys, the entire value should be enclosed in quotes. Channel program-specific settings can also appear on the Program tab: these are described in the appropriate sections below.

4.2.3 Tables

Intable : A table associated with the inbound part of this channel.

Outtable : A table associated with the outbound part of this channel. This is used by the channel program, in a channel specific way. Not all channels require this.

4.2.4 Inbound

The contents of the Inbound tab will vary depending on the type of channel being configured, so some of the fields listed below may not be present for a specific channel instance. The whole tab may be absent if there are no fields which are relevant to the channel being edited.

MTA Name : Used in P1 and P3 Simple Bind requests and responses, when using Directory-based configuration. If not set, will default to the value configured for the Switch as a whole. Not stored in the *mtatailor.tai* file. For SMTP channels, this value is used by other Isode MTAs which are reading this Directory entry when working out how to connect to this SMTP inbound channel: the MTA Name value is subsequently looked up in DNS.

Global Domain Identifier : This is used when verifying a Strong Authentication token, if used.

Command : The executable which provides the inbound component of the channel. If a relative path (or no path at all) is specified, it is taken to be relative to *(EXECDIR)*. Not stored in the *mtatailor.tai* file.

Presentation Address : This is the OSI address on which the inbound component of the channel (if any) will listen. Required only for P1, P3 and ACP142 channels. Not stored in the *mtatailor.tai* file.

Calling Presentation Address: This field allows the Transport, Session and Presentation selectors which are included in the Transport, Session and Presentation connect packets when opening an OSI connection to be specified. If a Network Address is specified and TCP/IP is being used as the Transport layer, then the Initiator will be bound to the specified IP address - this is intended to allow a specific interface to be chosen in a situation where a the Switch system has multiple interfaces. Specification of an invalid Network Address will thus prevent any outbound connections from being established (as the ‘bind’ operation will fail). This information is not stored in the *mtatailor.tai* file.

Application Context: Specifies the protocols which this channel provides to external MTAs. This field is only displayed for X.400 P1 channels, where there is a choice of P1

variants which can be provided. Note that this information is not stored in the *mtatailor.tai* file.

Reroute to another MTA: This field is only displayed for X.400 P1 channels. It allows all messages queued on the channel to be rerouted to a specific Peer MTA/Channel combination. This information is not stored in the *mtatailor.tai* file.

Associated Domain : For Internet Routing, gives the hostname to connect to when attempting to transfer messages to this MTA.

MTA Password : The password used in P3 Bind requests and responses. Not stored in the *mtatailor.tai* file.

4.2.5 Auth

The Auth tab is only present for OSI protocol channels. Its contents vary depending on whether a P3 or P1 channel is being configured.

P3 Initiator Authentication Requirements : These are the checks which are carried out by the P3 Initiator on the Responder's credentials. It is possible to specify that the MTA Name must be present, the Application Entity Title (AET) is present, the AET is valid, the Network Address is correct, that Simple Authentication succeeds, that Strong Authentication succeeds and that a Bilateral Agreement is present for the Initiator/Responder pair. Not stored in the *mtatailor.tai* file.

P3 Inititator Authentication Requirements : These are the checks which are carried out by the P3 Responder on the Initiator's credentials. Not stored in the *mtatailor.tai* file.

P7 Message Store : This can be set to be the Distinguished Name of a specific Shared Message Store, and if set in this way will prevent the P3 channel from being used to deliver into other Message Stores. Not stored in the *mtatailor.tai* file.

P1 Initiator Authentication Requirements : These are the checks which are carried out by the P1 Initiator on the Responder’s credentials. Not stored in the *mtatailor.tai* file.

P1 Responder Authentication Requirements : These are the checks which are carried out by the P1 Responder on the Initiator’s credentials. Not stored in the *mtatailor.tai* file.

Initiator RTS Credentials : These are the credentials to be used in a P1 bind request when connecting to another MTA. Both the MTA Name and Password can be specified: if the MTA Name is not set, it defaults to the MTA Name configured for the Switch as a whole. Not stored in the *mtatailor.tai* file.

Responder RTS Credentials : These are the credentials to be used in a P1 bind response when this MTA responds to another MTA’s bind request. Both the MTA Name and Password can be specified: if the MTA Name is not set, it defaults to the MTA Name configured for the Switch as a whole. Not stored in the *mtatailor.tai* file.

4.2.6 RTSE

The RTSE tab is only present for X400 P1 channels, and allows configuration of parameters specific to Reliable Transfer. This is described in detail in M-Switch Administration Guide.

4.2.7 MTA Links

The MTA Links tab is only present for X400 P1 channels.

MTA Name Links : This provides the mapping between the MTA Name in an incoming Bind Request and the entry in the Directory which holds the configuration information for the calling MTA. If a Bind Request arrives with an MTA Name which is not present in this table, then the bind will be rejected. After adding a new X.400 MTA to your

configuration (whether a local Tailoring MTA or an External MTA), you will need to make sure that your Links tables are updated.

4.2.8 ACP142 In, Out and Param Tabs

These tabs are present only for ACP142 channels, and are described in M-Switch Administration Guide.

4.2.9 Advanced

The Advanced tab is used for general configuration items common to many channels. Emitted formats: The type of addresses which are generated by this channel. Corresponds to the `adr` keyword in the *mtatailor.tai* file. The value is one of those below:

Value	Meaning
X.400	X.400 addressing
822	rfc822 addressing
any	either of the above types

Boundary-ACK : For outbound channels, controls acknowledgement generation and requests.

Bodyparts in : A list of X.400 EITs (up to a maximum of 28) that the channel will accept. Should be left empty for non-X.400 channels. Corresponds to the `bptin` keyword in the *mtatailor.tai* file.

Bodyparts out : A list of X.400 EITs (up to a maximum of 28) that the channel will output. Should be left empty for non-X.400 channels. Corresponds to the `bptout` keyword in the *mtatailor.tai* file.

Check for Bad Sender : This can be set to one of the following keys to control the policy for unroutable or unreplyable sender addresses:

Value	Meaning
reject	Generate delivery report (default)
accept	Artificially route on failure.

It is strongly recommended that the default policy, `reject`, is used. This will fail messages that arrive with an unroutable or unreplyable sender specification. If this is overridden, messages can easily be lost if any failure occurs.

If the `accept` mode is in effect, the MTA will attempt to route failure messages for unroutable senders either via the inbound MTA or if that fails to postmaster. This configuration item corresponds to the `bad-sender-policy` key in the *mtatailor.tai* file.

Check for Bad Message: This sets the checking mode of the channel. Normally this should be left as the default, which is strict checking, i.e. `reject`. However, for unusual cases it may be required to relax some of the constraints normally imposed. Setting the mode to `accept` will reduce the strictness of the checking to some extent. In particular it will allow RFC 822 messages with no, or multiple Date fields to be accepted. It has no effect for X.400 messages. Corresponds to the `check` keyword in the *mtatailor.tai* file.

Content In: This should only be set for shaper and checker channels, and specifies the types of content which the channel can handle. Corresponds to the `content-in` *mtatailor.tai* keyword. Common values are:

Value	Meaning
p2	A content of the form X.400 (84) IPM

p22	A content of the form X.400 (88) IPM
822	A content suitable for transfer over an RFC-822 protocol (i.e. one header and one text bodypart)

Content Out: Content out tailoring only applies to outbound channels, and lists the content types the channel can transfer. Available values are the same as for the **Content In** field. If left blank (i.e. set to none) this indicates that the channel is content independent. Corresponds to the `content-out` *mtatailor.tai* keyword.

Channel Specific Variables : This allows variables with arbitrary names and values to be set for an individual channel.

`add-prio-qualifier` : Variable applies to inbound channels. It is used to add the military messaging qualifier to the message envelope, with the value 'high'. If the qualifier is already present (with either value) then no action is taken. The value for the variable is a comma-separated list of the names of the priority levels for which this action should be taken. E.g.

```
add-prio-qualifier=normal,non-urgent
```

The other variables available to each channel executable are documented later in this section of the manual.

Connection Hold Time : Configures the length of time (in seconds) for which protocol connections are held open by outbound channels when there are no messages to transfer or deliver for the connected Peer MTA. Corresponds to the `conholdtime` *mtatailor.tai* keyword.

Domain Normalization: The amount of domain normalization of MTS addresses on inbound channels. Corresponds to the `domain-norm` *mtatailor.tai* keyword. The value is one of those below (the default is partial):

Value	Meaning
full	all domains in an MTS address are normalized
partial	only the next hop of an MTS address is normalized

With partial local domains are recognized and skipped, so the first non-local domain will be normalized.

Subtype in: This should be set for inbound channels only. Corresponds to the `subtype-in` keyword. Configures a content subtype which is used to control content conversion.

Subtype out: This should be set for outbound channels only. Corresponds to the `subtype-out` keyword. Configures a content subtype which is used to control content conversion.

Lookup policy for inbound messages: The lookup policies to use for messages coming in by this channel. It can take any of combination of the values specified in Lookup Policies. Corresponds to the lookup tailoring key.

If the optional prefix is specified in any of the policies, it will be prepended to the standard tables: `domain, or, channel, aliases` and `users`. For example, if the policy is `table=eg` then the tables that may referenced when evaluating that policy are `eg-domain, eg-or, eg-channel, eg-aliases` and `eg-users`.

Lookup timeout: The time in seconds to spend on non-table lookups. Corresponds to the `timeout` tailoring key. The default value is 30 seconds.

Maximum channel processes: This is the maximum number of instances of the channel the Queue Manager is allowed to run at any time. A value of 0 (the default) indicates that there is no maximum. Corresponds to the `maxproc` tailoring key.

MTA: A destination MTA for this channel. If this is set, all messages will be delivered to this MTA regardless of the destination MTA given in the message. This is useful for relaying all messages for a given channel via another MTA.

Maximum inbound connections: The maximum number of inbound connections to (instances of) this channel which are to be permitted. Corresponds to the `maxinconn` tailoring variable. The default is to allow unlimited inbound connections.

Maximum outbound connections: The maximum number of outbound connections from instances of this channel which are to be permitted. Corresponds to the `maxoutconn` tailoring variable. The default is to allow unlimited outbound connections.

MTA Report Request: Controls the setting of the mta report request for recipients. If the value set here is ‘less’ than the user report request, then the mta report request is increased to match the user request. Valid values are:

- basic : non-delivery reports requested
- confirmed : delivery reports requested
- audit-&-confirmed : delivery reports and subject trace information requested

Sort Key (primary): Sort keys are used to group "like" messages in order to make message processing more efficient. The primary sort key must be mta, user or none. The value none should only be used as a primary sort key for delivery channels.

Sort Key (secondary): Once the Primary Sort Key has been used to group messages, the Secondary Sort Key can be used to sort within these group. The values for this field can be time, size or none. For an outbound channel, the Primary Sort Key of "" "none" and "mta" are basically the same. This has been kept for historical reasons.

Within the *mtatailor.tai* file, the Primary and Secondary sort key values are combined in a single tailoring entry of

```
sort="<primary key>[ <secondary key>]"
```

By default, all outbound channels are sorted by mta only. Local channels should be sorted by user . Reformatters and other channels are normally sorted by none . However, setting a reformatter to mta or user allows multiple instances of the same channel to run concurrently.

4.2.10 Channel Specific Configuration

4.2.10.1 X.400 P1 Channel

The X.400 P1 channel uses the same program, `x400p1`, for both inbound (responder) and outbound (initiator) transfers. In addition, two-way alternate mode of operation can be configured, allowing the channel to operate as both sender and receiver over a single association.

The X.400 P1 channel can be started in different ways and with different options depending on the mode of operation required. These are summarized in the following subsections. Starting the channel for testing or debugging purposes in covered in Non-standard Use of the X.400 Channel. An example of an X.400 channel entry in the mtatailor.tai file is shown below.

Example of X.400 channel entry in mtatailor.tai file

```
chan X.40088 show="X.400 (1988)", type=both, adr=X.400,
name=X.40088, key=X.400in88, intable=X.400in88,
prog="x400p1 -i -te -fx400p1i",
content-out="p2,p22",hdrout="p22,p2,ipn",
outtable=X.400out88,probe=y,
appcont="2.6.0.1.6",outlookup="table",
inlookup="table",
bptout="ia5,g3fax,external,bilateral,
undefined,ttx,videotex,national,
encrypted,tif1"
```

The following fields specific to the X.400 P1 channel appear on MConsole's Program tab.

Allow P1 binds with invalid X.509 Subject DNs: corresponds to the `x509_allow_inv_aet` tailoring variable. When performing validation during Strong Authentication, the AET of the other party is normally checked against the Subject in their certificate. If this parameter is set to 'yes', the check is disabled.

X.509 Parent Directory: corresponds to the `x509_id` tailoring variable. Configures the directory path where the channel looks for its security environment when initializing prior to the use of Strong Authentication. If unset it defaults to the same directory as the Switch's tables.

Name of PKCS12 file: corresponds to the `x509_x400p1_p12_fname` tailoring variable. Configures the full directory path of the channel's P12 file.

Name of the passphrase file (to access the PKCS#12) : corresponds to the `x509_pphr` tailoring variable. Configures the passphrase needed to decrypt the PKCS12 file.

Directory in which trusted CA certificates are held: corresponds to the `x509_x400p1_trusted_ca_dir` tailoring variable.

4.2.10.1.1 Configuring the X.400 channel for Initiator mode

The channel is started by the Queue Manager using the values set in the `prog` field of the channel entry in the *mtatailor.tai* file. The values set may include the following:

```
x400p1 -i [-te|d] [-r|-s] [-f<logname>]
```

-i: Start as initiator. This value is mandatory.

-t <suboption> : The value of suboption can be either `e` to enable, or `d` to disable TWA on all connections. The value given here will override the value of the mode field in the channel table for the MTA. If not set the value in the channel table is used.

-r: Disable use of checkpointing and recovery facilities. If this option is not specified, the channel will attempt to resume the transfer of a previously aborted message.

-s: Enable saving of checkpoint data as the message transfer proceeds. The `-s` flag provides additional protection in the case of a system crash, as the necessary checkpoint data will always be saved.

-f <logname> : use the value logname when logging. If not specified, the program name `x400p1` is used by default.

4.2.10.1.2 RTSE Tab in MConsole

MConsole's RTSE tab allows various aspects of the way in which the X.400 channel connects to other MTAs to be configured.

The initiator settings are used by the channel when initiating a connection to another MTA; the responder settings are used by the channel when responding to an incoming connection from another MTA.

Mode: Two-way alternate (TWA) means that messages are sent and received on the same connection. Monologue means that once connected, messages are sent only one way: from the Initiator to the Responder.

Window: This is the number of checkpoints to go through before an acknowledgement is sent. If there is a failure in sending information across a network, the connection will be able to resend from the last checkpoint.

Checkpoint: This defines the number of bytes in a ‘checkpoint’.

RTS Transfer Timeout: The value is an integer, whose units are seconds per kilobyte, i.e. the actual timeout used is based on the size of the message. In this context, message size will include the surrounding layers of protocol information, and does not simply reflect the size of the message content and envelope. The default setting is zero, which implies waiting forever for a response. If message transfer fails because of the timeout, then the message will be tried again.

Note that you may also wish to set the `rtse_rec_timer` Isode variable at the same time.

X.400 Tracing: the style of tracing information to include in the envelope of messages transferred out by the channel.

Options are:

- ‘All’, meaning that all trace elements are included
- ‘ADMD’, which only includes trace elements relating to transfer between ADMDs
- ‘NIST’, which is as for ‘ADMD’, but with the domain identifier set from the MTSIdentifier of the message
- ‘No internal’, which means that no internal trace elements are included.
- ‘Local internal’, which means that only internal trace information elements corresponding to the global domain of the MTA are included.
- 'First element only', which means that only the first trace information element is included, and none of the internal trace information elements are included.

OR-Address Downgrade: This causes ORAddresses to be downgraded according to ISO/IEC DISP 12072–1 Annex C.

Use of rtse_rec_timer Variable

The `rtse_rec_timer` Isode variable controls the timeout which will be applied when waiting for the final acknowledgement from a X.400 P1 responder when a message transfer has been made. This is in addition to any RTS Transfer Timeout value; however the `rtse_rec_timer` value will only be used if an RTS Transfer Timeout has been set.

X.400 Bilateral Agreements

A Bilateral Agreement describes a set of overrides for a specific pair of X.400 MTAs which override the default values held in each of their individual entries. They are described in detail in the M-Switch Administration Guide.

MTA Links

With a Directory-based configuration, ‘MTA Links’ provide the mapping between the MTA Name field in an incoming P1 Bind Request and the Directory entry which contains the (local) configuration of information about the remote MTA. This is described in detail in the M-Switch Administration Guide.

X.400 Permanent and Scheduled Associations

These are described in detail in the M-Switch Administration Guide

X.400 Associations Limited by Priority

This is described in detail in the M-Switch Administration Guide.

X.400 Differentiated Service Code Point (DSCP)

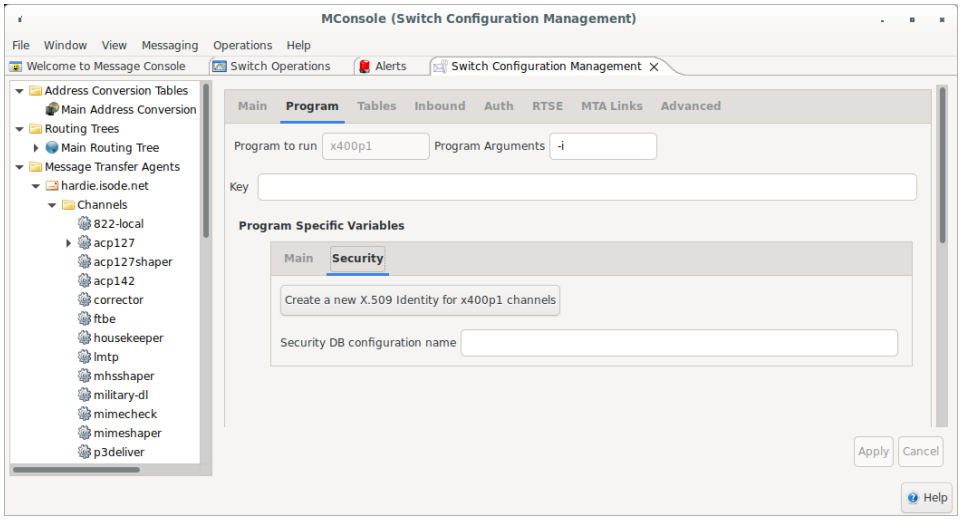
This is described in detail in the M-Switch Administration Guide.

4.2.10.1.3 Strong Authentication

Strong Authentication is used by the X.400 P1 Initiator if the Peer MTA to which connection is being attempted includes **Strong Authentication** in its **Authentication Requirements**. **Simple Authentication** and **MTA Name Present** can also be selected – the P1 Initiator will attempt strong authentication first if both strong and simple authentication are selected.

You can now configure the security environment for the X.400 P1 channel. This is shown in the diagram below, and described after that.

Figure 4.1. Configuring the X.400 P1 security environment.



There are five variables which can be configured. Some of these are mutually exclusive. You can configure the security environment by specifying a Digital Identity and a directory which contains trust anchors. Alternatively there is an obsolescent configuration in which a directory is specified which contains an X509 subdirectory which contains Digital Identities and trust anchors. The first Digital Identity found whose subject matches the DN of the channel is used. This latter method of specifying the security environment is deprecated and will be withdrawn in future releases.

Allow P1 Binds with Invalid X.509 Subject DNs

The default of **No** means that Strong Binds using an X.509 Certificate whose Subject DN does not match the DN in the Bind, are not allowed.

X.509 Parent Directory [Obsolescent and deprecated]

X509 Security Environment: this must be the name of a directory which contains a subdirectory named x509. This subdirectory must contain the X.509 PKCS#12 files which contain the digital identity to be used to generate the credentials used in a strong bind.

The Digital Identity used is the first one found which matches the DN of the entry of the X.400 P1 channel in the Directory which can be opened using the passphrase.

If not set, the default used is:

```
/etc/isode/switch
  C:/isode/etc/switch
```

These examples give the default values in the absence of this channel specific variable.

Unless the Digital Identity contain an unprotected private key you must configure the X.509 Digital Identity passphrase. The passphrase must be in a file in the same directory as the PKCS#12 file. The name of the file which must contain the passphrase is *<p12filename>.p12.pphr*

Warning: You should use the other values to configure the security environment. Use of this feature will be removed in a future release.

Name of PKCS12 file
Enter the pathname of a Digital Identity (PKCS12 file) to be used for constructing strong credentials.

Directory in which trusted CA certificates are held
The name of a directory which contains DER files to be used as trust anchors.

Name of the passphrase file (to access the PKCS#12)
This specifies the file that contains the passphrase needed to access the private key in the PKCS#12 file. If this is not configured, a default file path is used.

Allow Invalid DNs in Bind
If you check **AET Valid** in the **Authentication Requirements** then the X.400 P1 channel will not only ensure that the AET in the bind is valid (by reading the DN to retrieve the configuration of the remote MTA) but also check that the subject DN in the X.509 certificate provided in the bind matches the AET. You can disable the latter check by selecting **Allow Invalid DNs in Bind** on the **Program** tab.

See section Generating digital identities for detailed instructions on how to create a suitable certificate for the X.400 P1 channel

4.2.10.1.3.1 Creating a CA Using Sodium CA

See the *M-Vault Administration Guide* to create the initial CA.

4.2.10.2 ACP142 Channel

The ACP142 channel is described in detail in the M-Switch Administration Guide, along with the contents of the "ACP142 In", "ACP142 Out" and "ACP142 Param" tabs in MConsole.

The channel uses LDAP to obtain information. It uses the same global LDAP connection information as is used for routing lookup. The PP variables used are as follows:

ldap_host: Configures the name of the host on which the LDAP server containing the Switch configuration is running. This will default to '`localhost`' if not set.

ldap_port: Configures the port on which the LDAP server is listening: defaults to '`389`'.

ldap_sasl_user: Configures the SASL userid to use when binding to the LDAP server

ldap_sasl_pass: Configures the SASL password to use when binding to the LDAP server

ldap_sasl_mech: Configures the SASL mechanism to use when binding to the LDAP server.

If none of the '`ldap_sasl`' variables are set, the channel will fall back to use of a simple Bind using the MTA's DAP User and DAP password settings.

4.2.10.3 SMTP Channel

The SMTP channel, `slmtp` , comes in two parts, an inbound and an outbound process.

SMTP Server

The SMTP server supports the following SMTP extensions:

Table 4.4. SMTP Channel Extensions

HELP	Help information [RFC 1869]
EXPN	Distribution list expansion (only for X.500 Directory based lists). [RFC 1869]
SIZE	Message size declaration [RFC 1870]
8BITMIME	8bit-MIME transport [RFC 1652]
PIPELINING	Client side pipelining [RFC 2197]
DSN	Delivery Status Notifications [RFC 1891]
ENHANCEDSTATUSCODES	Enhanced mail system status codes [RFC 1893]
AUTH	Authentication [RFC 2554]

The server program for inbound messages makes use of the following channel-specific tailoring variables:

4.2.10.3.1 In Tab

When Return of Contents is absent, return: allows you to choose whether an absent ESMTP RET causes ROC (return of contents) to be set when gatewaying the message to an X.400 network. This corresponds to the `absent_ret_roc_full` channel-specific variable.

Maximum message size: allows configuration of the maximum acceptable message size (in bytes) for remote submission. It is used in the SMTP dialog with hosts supporting the SMTP extensions. This field corresponds to the `amms` channel-specific variable.

Allow binary data: configures the channel to accept binary data. Note that the resulting message will probably be changed. Such a message may also cause problems in processing. Corresponds to the `binary` channel-specific variable.

Block all connection attempts: If set, will cause the channel to always block connections. Corresponds to the `block` channel-specific variable.

Allow IP addresses with invalid hostnames: if set, then any connection will be allowed. If this is not the case, connections are only supported from hosts that can be reverse translated (the IP address, for example, can be converted back to a host name). Some people consider that setting `noname=false` is a violation of RFC 1123 (Internet host requirements). Corresponds to the `noname` channel-specific variable.

Listening address: by default, the SMTP server will listen on all available addresses, but can be restricted to a specific address via this configuration option; this might be useful in the case where the MTA is running on a system which has multiple network interfaces, for example. Corresponds to the `listen_addr` channel-specific variable.

Listening port: specifies the port on which the SMTP server is to listen. If not set, defaults to the standard SMTP port (25). Corresponds to the `listen_port` channel-specific variable.

Note on 8bit and binary data

The MIME specifications [RFC 2045] make a clear distinction between 8bit data and binary data. The former can include data with the 8th bit set (byte values 128 to 255). However, it cannot include the NUL character, nor Carriage Return or Linefeed, except as a CR LF

end of line pair. Also, 8bit data is still subject to the SMTP line length restriction of no more than 998 bytes between CR LF pairs.

This makes 8bit data unsuitable for the transfer of arbitrary binary data.

The SMTP server prevents the transfer in of binary data, unless the ‘binary’ option is set (the line length restriction is not applied). As a result of the way messages are handled within the Message Switch, data that violates the 8bit constraints may be changed, and also cause problems with operation of the Message Switch.

Note that 8bit data is only permitted within MIME body parts. It is not permitted to have non-ASCII characters within message or body headers. This is because there is no mechanism for assigning a character set to such characters. The same byte value corresponds to different characters in different character sets. Note that MIME provides a mechanism for encoding non-ASCII characters within heading fields [RFC 2047]. The presence of 8bit characters in message headers can cause problems in the operation of the Message Switch.

4.2.10.3.2 Out Tab

LMTP socket: This is only relevant for LMTP channels - see below.

Port number: Set the TCP port to be called (duplicates the -p command line flag). The corresponding channel-specific variable is `port=<integer>`.

Encode: Control how messages are sent. Values of none and default may result in sending messages which are invalid. Corresponds to the `encode=<value>` channel-specific variable. The table below describes valid encode value and the resulting action be the SMTP sender.

default	If the body part is mime-unknown and the receiving host indicates support for the 8BITMIME ESMTP extension then the message will be marked as BODY=8BITMIME, otherwise an attempt will be made to downgrade the content to a 7-bit transfer encoding if necessary.Any invalid binary data in headers or content parts will be passed as is. This setting does its best to send only valid messages but will not refuse to transmit a message which it cannot fix.
none	No conversion of content will occur on transmission. If the body part is mime-unknown and the receiving host indicates support for the 8BITMIME ESMTP extension the message will be marked as BODY=8BITMIME. Any invalid binary data in headers or content parts will be passed as is. 8-bit MIME content may be sent to hosts which do not support it. This setting does not alter a message in transit but may send messages which are invalid.
strict	If the body part is mime-unknown and the receiving host indicates support for the 8BITMIME ESMTP extension then the message will be marked as BODY=8BITMIME, otherwise an attempt will be made to downgrade the content to a 7-bit transfer encoding if necessary. Any invalid binary data in headers or content parts will cause non-delivery of the message. This setting will not send invalid messages. It will attempt to downgrade messages with 8-bit content to a 7-bit transfer encoding.

Maximum line length: Set a limit on the length of line that is sent. SMTP and RFC 1652 state that no more than 998 characters should be sent before a line break. By default there is no limit on the line length. Corresponds to the `line=<integer>` channel-specific variable.

Don't attempt Extended SMTP: do not attempt to use SMTP extensions. Corresponds to the `noesmtp` channel-specific variable.

Don't do MX record lookup: Do not use MX records. Corresponds to the `nomx` channel-specific variable and duplicates the -m command line flag.

Always generate relay DSN: Generate "relayed" DSN on transfer to DSN aware server. Even if a remote system advertises the DSN SMTP extension, setting this option will cause a "relayed" DSN to be generated for a message requesting a SUCCESS DSN on transfer to that system. Corresponds to the `relaydsn` channel-specific variable.

4.2.10.3.3 Errors Tab

The Errors tab configures how the inbound channel behaves when invalid recipient addresses are specified or when a very large number of recipients (often an indicator of unsolicited email) are specified:

Maximum number of errors on address commands: If present, sets a maximum number of errors which will be allowed on address-related commands (MAIL, RCPT, VRFY) before the command will no longer be recognized. Corresponds to the `maxerr` channel-specific variable.

Initial delay on errors: If present, sets a delay period (in seconds) which will be imposed after each address-related error. Corresponds to the `errdelay` channel-specific variable.

Maximum number of recipients before sending a [TEMP error]: If present, sets a maximum number of recipient addresses which will be accepted for an individual message. Addresses in excess of this maximum figure will be rejected with a temporary error. The presence of large numbers of recipient addresses in a message arriving from an external MTA is often an indicator of junk mail. Corresponds to the `maxrecips` channel-specific variable.

Maximum number of recipients before sending a [PERM error]: As for `maxrecips`, but will cause a permanent error to be generated. Corresponds to the `reciplimit` channel-specific variable.

4.2.10.3.4 Anti-Spam Tab

The Anti-Spam tab allows various settings associated with detection of unwanted email to be configured.

Real time blacklists (RBL): If this is set, it enables the Realtime Blackhole List (RBL) feature, see: http://mail-abuse.org/rbl

This corresponds to the `rbl` channel-specific variable.

A specific domain can be specified, which is used as a suffix to the calling IP address, for use with local implementation. An alternative target address can also be specified, as some RBL domains use non-standard addresses. Multiple RBL domains can be specified, in a semicolon-separated list (they are used by the SMTP inbound channel in the order in which they are specified). The syntax of the switch is thus:

```
rbl=<rbl_domain>["+"<target_address>][";"<rbl_domain>…]
```

Don't refuse connection on a DNSBL match: If this is set, messages from remote MTAs which are found on the configured Realtime Blackhole List are not rejected, but instead have their headers annotated with "`X-RBL-FOUND: <name> (<addr>)`", where `<name>` and `<addr>` are the name and IP address of the sending system. This corresponds to the `rblheader` channel-specific variable.

Reject code (SMTP): This selects the error code to be used when connections are rejected (for example, for the RBL or if `sloppy` is not `true` and there is no domain associated with the calling IP address). It can be set to `421` for a temporary reject (the default), or `553` for a permanent reject. It corresponds to the `reject` channel-specific variable. Additional information will be added to this, depending on the type of the reject.

SPF: SPF (Sender Policy Framework) is a way of checking that the purported sender address (the argument to the SMTP "MAIL FROM" command) is permitted to send mail

from the calling IP address. It is based on the use of special DNS records. For more information on SPF, see RFC 4408. The value of the field in MConsole controls the behaviour for different SPF results. The default behavior is to insert an extra heading field indicating the result. The value can be a comma separated list of result strings. If the result matches one of these, then the command is rejected. Possible result values are `fail`, `soft`, `temp` and `perm`. This control corresponds to the `spf` channel-specific variable.

4.2.10.3.5 Auth Tab

The SMTP inbound channel supports the AUTH keyword [RFC 2554]. This allows connections to the SMTP server to be authenticated. Various SASL authentication mechanisms are supported (as detailed below).

The Simple Authentication and Security Layer (SASL) provides a method for adding authentication support with an optional security layer to connection-based protocols. It also describes a structure for authentication mechanisms. The result is an abstraction layer between protocols and authentication mechanisms such that any SASL-compatible authentication mechanism can be used with any SASL-compatible protocol. See for more information.

The successful completion of an authentication exchange can cause the SMTP server to select a new incoming channel can be selected, allowing alternative authorization to occur. In order to select a new channel, a new mta name is constructed in the form `"<authorization_id>.auth"`. The SMTP server then reinitializes with this new MTA name, which may cause a different inbound channel to be selected. The simplest configuration which would allow this can be illustrated by the *mtatailor* fragment below:

```
chan smtp-external type=both name=smtp-external
prog=slmtp key=smtp show="with SMTP (external)"
content-out="822"
check=sloppy outadr="822" appcont="1.3.6.1.4.1.453.5.2"
bptout="ia5, mime-unknown" hdrout="822"
```

The example above assumes that "`isode.com`" is the MTA's local domain. An SMTP connection attempt coming in from the outside world would initially use the `smtp-external` channel (since its hostname is not listed in the `localhosts` table).

The successful completion of the authentication exchange results in the MTA name associated with the connection being changed. It has the form `<authorization_id>.auth` . This name is used for authorization, i.e. in looking up an entry in the `auth.mta` table. The association of the connection with a channel is also recalculated, through matching entries in the associated `mtatable`. The channel choice also affects authorization, and it affects other configurable behaviour.

A number of authentication-specific options are configurable for the SMTP channel via MConsole's Auth tab:

Disable AUTH command: Selecting this option prevents the SMTP server from advertising the AUTH command in its EHLO response, and causes it to reject AUTH commands with a "`502 5.5.2 command not implemented`" error. Corresponds to the `disable_auth` channel-specific variable.

When the option is not specified, it defaults to true (i.e. AUTH command is supported). When the option is specified, but no value given - it defaults to true as well.

Require authentication: This option specifies whether authentication is required before starting a mail transaction. If its value is true, and the client hasn't authenticated, the server will reply "530 5.5.1 (Must authenticate first)". Corresponds to the require_auth channel-specific variable.

When the option is not specified, it defaults to false. When the option is specified, but no value given, it defaults to true. See also "`soft_noauth`" option.

Soft NoAuth: If this option is true, all SMTP authentication 5XX error codes will be reported as 4XX error codes instead. Corresponds to the `soft_noauth` channel-specific variable.

BURL Command: This option specifies whether the BURL command is enabled on the server. The value "`auth`" says that BURL is advertised, but only allowed after successful authentication. If the client is not authenticated the server will respond "`530 5.5.1`" (Must authenticate first). Corresponds to the burl channel-specific variable, which can have values "`yes`", "`no`" or "`auth`".

When the option is not specified, it defaults to true (i.e. BURL command is supported). When the option is specified, but no value given - it defaults to true as well. See also the "`soft_noauth`" option.

EXPN Command: This option specifies whether the EXPN command is enabled on the server. The value "`auth`" says that EXPN is advertised, but only allowed after successful authentication. If the client is not authenticated the server will respond "`530 5.5.1`" (Must authenticate first). Corresponds to the `expn` channel-specific variable, which can have values "`yes`", "`no`" or "`auth`".

When the option is not specified, it defaults to true (i.e. EXPN command is supported). When the option is specified, but no value given - it defaults to true as well. See also the "`soft_noauth`" option.

VRFY Command: This option specifies whether the VRFY command is enabled on the server. The value "`auth`" says that `VRFY` is advertised, but only allowed after successful authentication. If the client is not authenticated the server will respond "`530 5.5.1`" (Must authenticate first). Corresponds to the `vrfy` channel-specific variable, which can have values "`yes`", "`no`" or "`auth`".

When the option is not specified, it defaults to true (i.e. VRFY command is supported). When the option is specified, but no value given, it defaults to true as well. See also the "`soft_noauth`" option.

Allow plaintext SASL: This option specifies whether plaintext SASL mechanisms (PLAIN, LOGIN) are allowed. If its value is false, those mechanisms will not be advertised in the EHLO response. When the option is not specified, it defaults to true. When the option is specified, but no value given, it defaults to true as well. When the option is set to "`tls`", plaintext SASL mechanisms are only allowed over a TLS-encrypted connection. Corresponds to the `allowplaintext` channel-specific variable.

Require TLS: This option specifies whether a TLS-encrypted connection is required before starting a mail transaction. If its value is true, and the connection is not encrypted, the server will reply "`530 5.5.1 A TLS-encrypted connection is required`". Corresponds to the `require_tls` channel-specific variable.

When the option is not specified, it defaults to false. When the option is specified, but no value given, it defaults to true.

SASL Mechanisms advertised: This option allows you to limit which mechanisms are advertised by the channel in its `EHLO` response. The intersection of the set of available mechanisms with this list is returned in the `EHLO` response: e.g. if "`PLAIN;DIGEST-MD5;GSSAPI`" are available and the value of this option is "`SRP;GSSAPI;DIGEST-MD5`", the `EHLO` response will list at most `DIGEST-MD5` and GSSAPI. The corresponding channel-specific variable is `sasl_mechs = <`";" separated list of SASL mechanisms>.

"At most", because other options like `allowplaintext` affect the final list of available options as well.

4.2.10.3.6 SASL Mechanisms

The Message Switch supports multiple SASL mechanisms via a plugin system. When the Switch starts up it loads all the plugins installed in *(LIBDIR)/sasl2*. This makes it simple to completely disable certain mechanisms (by removing the plugin file and restarting the Switch) or to add additional mechanisms (by copying in the new plugin and restarting the Switch).

Each mechanism supplied has different characteristics that might make it more or less useful for a given Message Switch.

Table 4.5. -SASL Mechanism Characteristics

Mechanism	Approach	Security
PLAINLOGIN	Sends plaintext passwords across the network.	Very weak
CRAM-MD5	Basic challenge/response, but vulnerable to server spoofing attacks	Weak
NTLM	Basic challenge/response, using a Microsoft-specific algorithm	Weak
DIGEST-MD5	Challenge/response	Good
OTP	One-time password	Good

4.2.10.3.7 TLS Tab

TLS support: Configures the outbound use of TLS by the SMTP Client and whether the inbound SMTP server supports TLS and advertises this support in the response to an `EHLO` command. Possible values are:

Value	Meaning
OPTIONAL	Use TLS if available
SERVER	TLS to be advertised by SMTP Server only
YES	Same as OPTIONAL

This option corresponds to the `tls` channel-specific variable.

Identity: Configures where to look for the digital identity to be used by the SMTP server. An identity is contained in a file which can be: *rsa.p12* (with passphrase file *rsa.p12.pphr*), *dsa.p12* (with passphrase file *dsa.p12.pphr*) or *id.p12* (with passphrase file *key.pphr*). The *id.p12* form is deprecated. This is a PKCS#12 file containing the private key and certificate. If this file is passphrase protected, the passphrase should be held in a text file. Corresponds to the `identity` channel-specific variable.

Some additional channel-specific variables can only be set via the Advanced Tab in MConsole at present:

local: More stringent checks to see if MX records point to the local MTA (duplicates the -l command line flag).

reconnect_retries=<integer>: For SMTP connections only, the number of times to retry the complete connection processing. Defaults to no retries.

SMTP Client

The SMTP client program for outbound messages understands various channel-specific variables, which can be set from MConsole.

-p port: It can be given this flag to tell it to connect to a *tcp* port other than the default. This can be set up by setting the tailor entry to something like the following:

```
chan smtp-odd prog="smtp -p 2001",show="Odd smtp"...
```

-l : This flag only makes sense if the DNS is being used. This makes more stringent checks on the DNS MX records to see if a host is actually the local host by comparing IP addresses as well as domain names.

-m : It can also be given this flag if compiled with `nameserver` support, in which case it will not resolve addresses using MX records. This is occasionally useful for internal traffic.

An 8bit message is one which is flagged as 8bit when transferred into the message switch from another switch, using the 8BITMIME SMTP extension [RFC 1652].

A message will also be flagged as 8bit if, when the `mimeflatten` channel processes a message, MIME body parts which have the 8bit or binary content transfer encoding are found.

When an 8bit message is transferred to a host that supports the 8BITMIME extensions, it is flagged as an 8bit message to that host, and transferred unchanged.

When the receiving host does not support the 8BITMIME extension, or the message is not flagged as 8bit, then the transfer method depends on the configuration of the encode option.

4.2.10.3.8 Greylist Tab

4.2.10.4 LMTP Channel

The LMTP channel is an outbound channel which uses the `slmtp` program. It is designed to enable the integration of the Message Switch with the IMAP Message Store.

It is configured in the same way as the smtp channel (See SMTP channel), with the following exceptions:

```
name lmtp
show "with LMTP"
prog slmtp
type out
access mts
lmtp <hostname> | <UNIX socket name>
port <port number>
```

The value of the *lmtp* and *port* tailoring variables depends on whether you are using TCP or UNIX sockets.

Using TCP

When using TCP to deliver, the `lmtp` variable should be set to the `hostname` of the system to which the channel is to connect. If no `hostname` is specified, `localhost` will be used instead. The TCP port to use will need to be configured as well. It should NOT be set to 25. In MConsole, these fields can be set on the Out tab for the channel: the LMTP socket field should be set to the target `hostname`.

Note also for TCP, that if you are delivering to a host which is not the local host, as defined in *loc_dom_mta* , you must specify the following in the channel configuration:

```
mta <hostname>
```

Identity: this field of the Out tab allows the directory containing the TLS identity information to be configured. If not specified, then the value of the tailoring variable `tls_path` (set on the MTA's Security tab) is used. If that is not set, then *(TBLDIR)/tls* will be used. It corresponds to the `identity` channel-specific variable.

Using UNIX sockets

When using UNIX sockets, the value of the `lmtp` variable (the LMTP socket field on the Out tab in MConsole) should be the name of a UNIX socket. For example, the UNIX socket name, for SMS 3.0, is */var/md/store/ipc/lmtpd*. The following example shows a typical configuration for the LMTP channel.

```
chan lmtp prog=slmtp, show="with LMTP", type=out, access=mts,
adr=822, content-out=822,
lmtp=<unix socket name>, hdrout=822-norm,
bptout="ia5, mime-unknown, mime-multipart-signed,
mime-multipart-encrypted"
```

4.2.10.5 P3 Channel

The P3 channel provides submission and delivery of X.400 messages using the P3 protocol, either directly to/from a P3 User Agent, or indirectly via the P7 Message Store. Message delivery and report delivery operations take place on associations initiated by either the MTA or MTS user, while message and probe submission operations are accepted on associations initiated by the MTS user. These actions implement the `mts-access` and `mts-forced-access` application contexts for P3.

Channel-specific configuration variables are:

ndr_on_error: Controls the behaviour of the P3 Channel when some errors are encountered during message delivery. If set to any value, the message concerned will be non-delivered: if unset (the default) a temporary error will be logged and message delivery will be retried later.

acceptall: Setting this to "true" switches on the accept all option. In this case the P3 Server will not reject messages containing a bad address and will instead cause a non delivery report to be generated. In MConsole, this can be configured using the Accept all recipients option on the Program tab.

P3 protocol submission and delivery channels are implemented via the `p3server` and `p3client` programs. Message delivery and report delivery operations take place on associations initiated by either the MTA or MTS user, while message and probe submission operations are accepted on associations initiated by the MTS user. These actions implement the `mts-access` and `mts-forced-access` application contexts for P3. As a result, a normal MTA configuration will have two P3 channels configured - a `p3server` channel which can perform submission and delivery, and a `p3deliver` channel which only performs delivery.

NOTE: If `tsapd` is to be used, as opposed to `iaed`, an entry of the following form must be added to the *(SHAREDIR)isoservices* file for the P3 submission channel:

```
tsap/p3 "403" /opt/isode/libexec/p3server
```

The P3 channel processes either use standard table-based methods or X.500 Directory lookup to obtain the information necessary to deliver messages or accept them for submission. The tailoring required differs between the two lookup methods.

If table-based lookup is used, *mtatailor.tai* file entries similar to the following entries are required:

```
tbl p3 show="P3 protocol submission & delivery"
chan p3 show="P3 submission & delivery" type=both, access=mts,
adr=X.400, key=p3server, prog=p3client,
sort="user priority time",content-out="p2,p22", outtable=p3,
hdrout="p2,p22,ipn",
```

```
bptout="ia5,g3fax,external,bilateral,undefined, ttx,
videotex,national,encrypted,tifl",
intable=p3
```

If X.500 Directory lookup is used, *mtatailor.tai* file entries similar to the following entries are required:

```
tbl p3 show="P3 protocol submission & delivery"
chan p3 show="P3 submission & delivery" type=both, access=mts,
adr=X.400, key=p3server, prog=p3client,
sort="user priority time",content-out="p2,p22", outtable=p3,
hdrout="p2,p22,ipn",
bptout="ia5,g3fax,external,bilateral,undefined, ttx,
videotex,national,encrypted,tifl"
```

Instead of MTA initiated delivery (using the `p3client` process), the P3 channel can be configured to deliver messages only when the MTS user indicates its presence and specifically requests delivery. For this the `prog` field needs to be removed (so that the Queue Manager does not start any P3 channel instances). When an MTSBind arrives at the P3 Server process, the requested application contexts in the Bind Argument are checked, and if the Message Delivery Service Element is indicated, delivery processing is turned on.

As messages may be in the system some time awaiting delivery, the message timeout value may need to be altered. This value is set in the *mtatailor.tai* file variable, `returntime`.

Changes to the *mtatailor.tai* file for MTS initiated delivery will take effect when the MTA is restarted.

4.2.10.6 List Channel

The List channel performs expansion of Distribution Lists. The channel executable can work in X.400 or Internet mode, but if you want to support both modes of operation on the same MTA you will need to configure two channel instances.

The Program tab for the List channel in MConsole has a number of controls:

Operational mode: Whether this channel is working in X.400 or Internet mode, or Default mode. In Default mode, the channel will use X.400 mode if the MTA is X.400-only and will use Internet mode otherwise. This single control sets the `x400` or `internet` channel-specific variables as appropriate.

Use Directory lookup: Whether this channel should use Directory lookup for DL expansion. Defaults to `"no"`, which means that the channel will use table-based expansion instead. This control sets the `dirlookup` channel-specific variable.

Expand sublists immediately: Controls whether entries in the Distribution List being expanded which are themselves the addresses of other Distribution Lists should be expanded immediately, or left for processing by another List channel instance after resubmission. This control sets the value of the `dosublists` channel-specific variable.

Allow empty lists: Controls whether messages which are addressed to an empty Distribution List cause a DR to be generated (empty=false) or marked as successfully processed (`empty=true`). This control sets the value of the empty channel-specific variable.

Local DIT Base: The value for this field is a string-encoded DN, giving the point in the DIT under which searches for Distribution List entries should be performed, if Directory lookup is being used. Sets the value of the `localdit` channel-specific variable.

Search depth: Controls how the search for Distribution List entries should be performed, if Directory lookup is being used. Sets the `search` channel-specific variable.

Ignore dl-expansion-prohibited: Allows the channel to be configured to ignore the `dl-expansion-prohibited` flag which may be present in a message, and which would normally cause the List channel to generate a negative DR for the message. Sets the `ignore-dl-exp-prohibited` channel-specific variable.

4.2.10.7 Checker Channel

The Checker channel performs anti-spam and anti-virus checking. Full details of this channel and how to configure it are provided in the Content Checking chapter of this manual and M-Switch Administration Guide.

Main Tab

Per-user LDAP config: Setting this to `"Yes"` will enable the channel to use per-user configuration read from the Directory. This control sets the `ldap_userconfig` channel-specific variable.

LDAP config prefix: This allows the a prefix to be added to the name of the Directory Profile used for per-user configuration. The control sets the value of the `ldap_config_prefix` channel-specific variable.

XML configuration file: Specifies an alternative XML configuration file for the channel (the default is “<channel-name>-config.xml”. Sets the value of the `configfile` channel-specific variable.

Tcl script used in init: This specifies a the name of a Tcl script which will be executed during the initialization phase of the Tcl-based component of the Checker channel. Sets the value of the `script` channel-specific variable.

Tcl command used in init: This specifies a Tcl command which will be executed during the initialization phase of the Tcl-based component of the Checker channel. Sets the value of the `command` channel-specific variable.

The following channel variables can be overridden by the channel configuration within the scanconfig setup for the channel:

Anti-virus Tab

virusengine: The name of the Tcl package to load which is the interface to the Anti-Virus engine. Not all platforms support all A-V engines. One of:

- VirusNone - no checking performed
- VirusClamAV
- VirusNorman
- VirusSophos

repair If true, attempt to repair infected attachments

nocheck If true, disables virus checking

uudecode If true, look for infected uuencoded attachments

maxsize If greater than zero, gives the maximum size in bytes for attachments which are checked

tempdir Temporary directory used to create files from attachments for A-V checking. Can be on a RAM disk for efficiency

nodelete Delays the deleting of temporary files (unless tempdir is used).

Before the checker channel can be used, it will be necessary to set up a minimum set of checking rules and build the cache file which holds the recompiled version of these rules.

This minimum set of rules is contained in the file *(SHAREDIR)/msgcheck.zip*. You must copy the file to *(ETCDIR)/switch* and unzip it, creating a *msgcheck* subdirectory. After doing this, you will need to run the `cachebuild` Tcl script for each checker channel which you have created:

```
cachebuild mimecheck
```

On Windows you will need to explicitly run the Tcl interpreter and pass in the name of the Tcl script (`cachebuild`) as the first command line argument.

Notes on AntiVirus Packages

ClamAV

M-Switch uses ClamAV by communicating with the daemon process included in the package. When configuring ClamAV (in the *clamd.conf* file), you will need to check and modify the following variables:

- **TCPSocket**: this needs to be uncommented and should be set to the default which M-Switch assumes of 3310
- **User**: this should be set to either the '`pp`' userid or '`root`', so that the clamd daemon can access files in the MTA's queue.

Norman

M-Switch uses a programmatic interface to the Norman Anti-Virus libraries, so simply installing the Norman packages should make the functionality available.

Sophos

M-Switch uses the Sophos AV package by interfacing to the SAVDI daemon over a protocol connection. The SAVDI daemon installs on top of the standard Sophos anti-virus product. An appropriate `SAV Dynamic Interface` installation package for your platform can be supplied on request by Isode Support. The daemon is configured via a file named *savdid.conf*.

On Unix platforms, the default settings in this file are correct, but on Windows it will be necessary to look for the "channel" section which configures the SSSP protocol and set the `allowscanfile` configuration variable to the value `FILE`.

On all platforms you will also need to ensure that the daemon process runs with sufficient privilege to be able to access files which have been created by the MTA processes.

Anti-Spam Controls

prionormal Rule priority level applying for normal level

priomild Rule priority level applying for mild level

prionone Rule priority level applying for no A-S.

scanall Scan all attachments, not just text bodies, for A-S.

no_unknown_charsets Enable a rule which traps unknown charsets

General controls

quarantinedir Directory for quarantined messages. This can contain fields in which are substituted date and time values. See the definition of the archive directory.

surbl Suffix to be used for anti-spam URL lookup. If set to the empty string, disable SURBL lookup. Standard rule files define this to be `multi.surbl.org`.

redirect Default address for redirection, if the action for a rule is to redirect.

showwords Display matched words in annotated messages.

scanlog If set, causes logging of various scanning events.

scancontext Size of context for matches shown in scanning logging.

Setting Up ClamAv on Windows for use with a Checker Channel

Instructions on how to install and configure ClamAv on Windows.

- Download ClamAV for Windows from `http://oss.netfarm.it/clamav`
- Create *C:\clamAV\db*
- Copy the contents of the downloaded files into `C:\clamAV`
- Import the registry settings from *C:\clamav\clamav.reg* by either double clicking or using the following command within an elevated command prompt:

```
regedit /s c:\clamAV\clamav.reg
```

- Install the Clam Service, running the following from an elevated command prompt (note the use of two --)

```
c:\clamAV\clamd.exe --install
```

- Change the start-up of this service using the following; note the white space after the equals sign

```
sc config "ClamD" start= auto
```

- Download the latest ClamAV updates from `http://www.clamwin.com/content/view/58/27/`, copy these two files to *C:\clamav\db* (This is an offline update - you can also use the FreshClam program to automate this).
- Start the clam daemon service

```
Net start "clamD"
```

4.2.10.8 CCCP Channel

The Content Checking and Conversion Protocol (CCCP) is a means for a checking channel to communicate with a server process which can perform both content checking, and also change the content. It can cause the message to be non-delivered, or deleted for each recipient, or the message can be redirected for a recipient. Full details of this channel and how to configure it are provided in M-Switch Administration Guide.

4.2.10.9 Shaper Channel

The Shaper channel is used to perform content conversion. It is documented in the Content Conversion section of this manual.

4.2.10.10 822-local Channel

The 822-local channel delivers Internet messages into a Unix maildrop file. There are four channel-specific variables which can be set from the Program tab in MConsole:

Mailbox name: Name of mailbox file. It defaults to `ppmailbox`. This control sets the `mboxname` channel-specific variable.

Mail filter: Name of mailfilter file. Defaults to `.mailfilter`. This control sets the `mailfilter` channel-specific variable.

Mailbox delimiter 1: Delimiter character string for start of message in mailbox. Defaults to "`\\1\\1\\1\\1\\n`". This control sets the `delim1` channel-specific variable.

Mailbox delimiter 2: Delimiter character string for end of message in mailbox. Defaults to "`\\1\\1\\1\\1\\n`".This control sets the `delim2` channel-specific variable.

4.2.10.11 Housekeeper Channel

The Housekeeper channel performs a variety of functions. Its primary purpose is to generate X.400 Delivery Reports and Internet DSNs when requested by other channels. It is also responsible for generating warnings when message delivery or transfer is delayed (in MIXER and Internet-only configurations), resubmitting messages when requested, reloading messages into the Queue Manager when they have been updated, deleting messages from the queue on command, redirecting messages to alternative recipients, converting X.400 DRs to Internet DSNs and tidying up general ‘trash’ in the MTA’s queue.

There are only two channel-specific tailoring variables:

Discard messages: If a message for which a DR or DSN is being generated has an empty originator address, there is nothing to address the DR or DSN to, and the DR or DSN would normally be discarded. To prevent this, the `discard` channel-specfic variable can either be set to `"no"`, in which the Postmaster address for the MTA will be used as the recipient of the DR/DSN, or it can be set to the Internet address of the desired alternative recipient of the DR/DSN.

Text body templates: This configures a template file used to configure the text which is included in DSNs and MDNs. If the filename is a relative filename, it is relative to *(ETCDIR)*. This control sets the `template` channel-specific variable.

Trash Facility

The *trash* facility within the Housekeeper channel frees storage which no longer is of interest to the mail system. It only removes files and directories which it deems suitable for deletion and whose latest modified times are suitably ancient. The time interval after which a file is suitably ancient may be configured via the "Trash Lifetime" item on the "Advanced" tab for the MTA in MConsole, or via an entry of the form "`set trash_lifetime=<interval>`" in the *mtatailor.tai* file.

If not specified, the interval is set to the default value of three days. If given, the interval must be specified as a string of the form `4d 2h` which represents an interval of four days and two hours. The time unit abbreviations recognised are:

- s seconds
- m minutes
- h hours
- d days
- w weeks

dr2dsn Facility

The *dr2dsn* facility within the Housekeeper channel converts X.400 Delivery Reports into NOTARY Delivery Status Notifications (DSNs) in the form of MIME RFC 822 messages. Once constructed these messages are then submitted to the Queue Manager for return to the sender. The text body generated for the DSN can be configured.

Report Generation Facility

The Housekeeper channel is responsible for generating X.400 Reports and NOTARY DSNs. It is scheduled when a channel decides that one or more recipients of a message require a report. The channel saves the information required to generate the report in the message queue, and returns an indication to the Queue Manager that a report is required.

An X.400 report may contain a delivery report indicating successful delivery to a recipient, or a non-delivery report indicating that the message could not be delivered to a recipient. A single X.400 report may also contain both types of delivery report for messages with multiple recipients; for example, a message could require an X.400 report that contains a delivery report for Recipient A, but a non-delivery report for Recipient B.

A NOTARY DSN may similarly indicate successful or unsuccessful delivery. It may also indicate distribution list expansion, relay to an MTA which does not support NOTARY, or that the message has been delayed. A single DSN may also contain multiple types of delivery report for messages with multiple recipients; for example a message could require a DSN that contains a delivery report for Recipient A, a non-delivery report for Recipient B, and a delayed report for Recipient C.

The text body generated for the DSN can be configured.

Discard Address Function

When the Housekeeper channel encounters an unroutable DSN, or a message with no return address causes a permanent error, the channel can discard the error or send a DSN to a chosen address. This is configured via the value of the `discard` channel-specific variable.

4.2.10.12 FAPI Channel

NOTE: The FAPI channel is only supported on Windows platforms.

The FAPI channel provides file-based message submission and delivery facilities. The channel is actually implemented as two independent parts: a `fapiserver` process which runs as a Windows Service, providing a submission service, and a `fapichannel` channel which provides the delivery service. The channel uses a private format, where elements of the X.400 P1 envelope and P22 content (e.g. addressing and some header fields) are contained in a textual ‘header’ file, with bodyparts supplied as separate files referenced from the associated header file.

A detailed specification of the FAPI interfaces can be obtained from the relevant ECB documentation. Information in this manual is confined to the details of how to configure and run the channel.

FAPI Submission Service

For normal operation, the FAPI submission service should be installed as a Windows service. This can be done using the Services tab of the Switch Operations View in MConsole (or the standalone Isode Services Manager). Select the “Add” option, and configure fields as shown below:

Service Name	isode.pp.fapi
Description	Isode M-Switch FAPI Server
Executable path	*(SBINDIR)/fapiserver.exe*
Service arguments	-c fapi

All other parameters can be left empty or with their default values.

Once the submission service is installed, you can start it using the Services tab of the Switch Operations View in MConsole (or the standalone Isode Services Manager) . If you wish to run the submission service as a standard program, you will need to specify the `"-d"`

(debug) command line flag, in addition to the mandatory channel name ("-c") command line flag and argument; i.e:

```
>C:
>cd \Program Files\Isode\bin
>fapiserver -d -c fapi
```

Once the submission service is running, all of the rest of its configuration information is read from the relevant channel entry in *mtatailor.tai* and either directory or table-based configuration. As the configuration information is shared between the submission and delivery processes, it is described separately (below).

4.2.10.13 FAPI Delivery Channel

The FAPI channel is a standard delivery channel. Its tailoring is described below.

FAPI Tailoring

Directory-based configuration of the FAPI channel and associated Points of Access (PoAs) relies on the fact that from a configuration standpoint a PoA looks very much like a P7 Message Store. A dummy Shared Message Store can be configured, allowing individual Message Store Users with associated OR-addresses and Mailbox Roots to be assigned to it. A 'real' P7 Message Store process is never run: instead the FAPI submission server and delivery channel assume roughly the same role as the P7 process's submission and delivery functions.

The following discussion assumes that you have already set up a standard X.400 Messaging Configuration (without a Message Store), as described in the M-Switch (X.400) Administration Guide.

Next, use MConsole to create a Message Store object. You can assign any name to the Message Store, and give it a Presentation Address of "NS+" (i.e. a NULL address). "Listen level" and "Invoke level" can both be set to "Static". You will need to associate the X.400 Routing Tree you should have already created with the Message Store.

Next, use MConsole to create a new channel, with the name 'fapi'. This should have the following settings:

```
type=both
access=mts
no outbound protocol selected
prog=fapichannel
key=fapi
store="<cn=MS, cn=Messaging Configuration…..>"
content-out="p2 p22"hdrout="p2 p22"sort=user
adr=X.400 bptout="ia5 undefined bilateral"
```

Any fields not mentioned above can be left with their default values. The values entered for the store key should be the Distinguished Name of the Message Store directory entry created earlier. This can be discovered by using the Sodium program to examine the section of the Directory Information Tree containing the Messaging Configuration - it will probably be something like: "cn=MS, cn=Messaging Configuration, ou=MHS, o=Example Corp, c=GB". Note that the DN needs to be enclosed in <> characters, and will need to be quoted if it contains any spaces.

Next you need to create an 'X.400 Message Store User' corresponding to each POA which you wish to support. Configure the MS User's "Mailbox Path" to be the full pathname to the user's Point of Access. You can select a zero-length password - it will not actually be used.

Once you have created an MS User with the EMMA wizard dialog, navigate to the user's properties, and change their `ppchannel` attribute value from "`p3`" to "`fapi`". This will cause the MTA to deliver messages to them using the FAPI channel.

You can now start the services which make up the Isode X.400 Message Switch, using the Isode Service Manager. Remember that the Isode MTA Tailoring Daemon service must start first.

You are now ready to test submission and delivery.

The complete list of channel-specific variables which may be set for the channel is:

inreadydir: The name of the inbound 'ready' subdirectory for a mailbox: defaults to "`in/ready`"

inheaderdir: The name of the inbound 'header' subdirectory for a mailbox: defaults to "`in/header`"

indatadir: The name of the inbound 'data' subdirectory for a mailbox: defaults to "`in/data`"

outreadydir: The name of the outbound 'ready' subdirectory for a mailbox: defaults to "`out/ready`"

outheaderdir: The name of the outbound 'header' subdirectory for a mailbox: defaults to "`out/header`"

outdatadir: The name of the outbound 'data' subdirectory for a mailbox: defaults to "`out/data`"

delrepheaderdir: The name of the delivery-report 'header' subdirectory for a mailbox: defaults to "`del_rep/header`"

delrepreadydir: The name of the delivery-report 'ready' subdirectory for a mailbox: defaults to "`del_rep/ready`"

subrepheaderdir: The name of the submission-report 'header' subdirectory for a mailbox: defaults to "`sub_rep/header`"

subrepreadydir: The name of the submission-report 'ready' subdirectory for a mailbox: defaults to "`sub_rep/ready`"

store: The DN of the FAPI Server's own entry in the Directory

password: The userPassword attribute associated with the FAPI Server's Directory entry

sloppy: if `sloppy` is set to "`true`" (the default is "`false`"), the inbound FAPI server will ignore (but log) any unknown keys or malformed lines in its input data, instead of reporting an error and generating a negative submission report (the default behaviour).

forcequote: if `forcequote` is set to "`false`" (the default is "`true`"), data values which are written to report and message files will only be enclosed in double-quotes if the data values include whitespace. The default behavior is to quote data values regardless of whether they need it or not.

4.2.10.14 P1 File Channel

The p1file channel comes in two parts: an inbound (P1 File Server) and an outbound process (P1 File Client). The processes read from and write to binary files containing the BER encoding of P1 messages. Channel-specific tailoring controls are:

4.2.10.14.1 In Tab

Error Directory: Configures where the inbound channel writes messages which cannot be submitted for some reason. The default value is */var/isode/p1file/errors* on Unix and *C:/Isode/p1file/errors* on Windows. Sets the value of the `errdir` channel-specific variable.

Input Directory: Configures where the inbound channel expects messages for submission to be placed. The default value is */var/isode/p1file/inbound* on Unix and *C:/Isode/p1file/inbound* on Windows. Note that on Unix, this directory must be readable/writeable by the PP user. Sets the value of the `indir` channel-specific variable.

Remote MTA: Configures the MTA name which the channel uses when submitting messages. This variable is mandatory unless the `p1file` channel program has been configured to run with the "`-m <mtaname>`" switch. Sets the value of the `remote_mta` channel-specific variable.

Sleep: The length of time for which the channel will sleep between processing one set of inbound messages and checking for a new set. Only used on Unix (on Windows a ChangeNotification mechanism is used instead). Defaults to 1 second. Sets the value of the `sleep` channel-specific variable.

Prefix: Defines a file prefix which is required for input files (i.e. files which do not have this prefix will be ignored). Can be either a fixed character string (e.g. "`msg.`") or "`%p`". In the latter case, the `p1file` process will expect all message files to start with a priority indicator character of "h", "m" or "l", indicating high, medium or low priority respectively. Messages will then be processed in priority order. The default is for all files to be processed. Sets the value of the `prefix` channel-specific variable.

Suffix: Defines a file suffix which is required for input files (i.e. files which do not have this suffix will be ignored). Must be a fixed character string (e.g. "`.msg`"). The default is for all files to be processed. Sets the value of the `suffix` channel-specific variable.

Number of open retries: On Windows, the ChangeNotification which alerts the p1server process to new files will fire when a new file is created in the input directory, but before the file has been closed by the process which is creating or copying it causing the p1server to attempt to open the file and discover that it is locked. When `num_open_retries` is set non-zero, the p1server will wait for a short interval and then attempt to open the file again, up to the configured limit on the number of attempts. Defaults to 0 (i.e. no retries). Sets the value of the `num_open_retries` channel-specific variable.

Retry timeout: The time (in seconds) to wait before retrying after a failed file open. Defaults to 1 second. Sets the value of the `retry_timeout` channel-specific variable.

Accept all recipients: Configures whether the channel should accept all recipients and subsequently generate negative DRs for any invalid addresses, or should reject a message which contains invalid addresses. Sets the value of the `acceptall` channel-specific variable.

4.2.10.14.2 Out Tab

Output Directory: Configures where the outbound channel writes outbound messages to. The default value is /var/isode/p1file/outbound on Unix and C:/Isode/p1file/outbound on Windows. Note that on unix, this directory must be readable/writeable by the PP user. If "%m" is specified in the path, it will be replaced by the name of the outbound MTA. Sets the value of the `outdir` channel-specific variable.

Out File: Configures a template for outbound file names. If used this must include "`%q`" somewhere - this will be replaced with the QueueIdentifier of each message written. The template may also include "`%p`" for standard priority and "`%P`" for military priority (including use of priority-level-qualifier if present). For standard priority, the letters n,l and h are used to indicate normal, low and high priority messages. Reports are indicated with an uppercase R. Probes are just set to low priority. For the "`%P`" substitution, letters `i,p,r,d,o` and `f` are used to indicate immediate, priority, routine, deferred, override and

flash priority levels. With the "%P" substitution, if the priority-level-qualifier envelope extension is not present, just `p,d` or `f` will be output. If not specified, the default name of output files is just the QueueIdentifier. Sets the value of the `outfile` channel-specific variable.

4.2.10.14.3 Table Based Configuration

A typical *mtatailor.tai* entry for the `p1file` channel would look like:

```
chan p1file type=both name=p1file prog=p1filechannel key=p1file
show="P1 file transfer channel" content-out="p2,p22"
outinfo="outdir=/var/isode/p1file/outbound/%m"
ininfo="remote_mta=p1file"
bptout="ia5, g3fax, ttx, videotex, national, encrypted, undefined,
       voice, tif0, bilateral, odif, iso6937, external, tif1"
hdrout="p2, p22" inadr="x400" outadr="x400"
contentin="p2, p22"
contentout="p2, p22"
```

Note that neither inbound nor outbound tables are required.

4.2.10.14.4 P1File Client

The outbound side of the p1file channel acts as a standard transfer-out channel. Messages and reports are written to output directories. The output directory paths may include a component which identifies the destination MTA, if required.

4.2.10.14.5 P1File Server

The p1file server process p1fileserver monitors a specific input directory for new files. When a file arrives in the input directory, the process reads the contents of the file and attempts to decode it as a P1 MTS APDU (i.e. an ASN.1 Sequence of envelope and content). If this is successful the resultant message is submitted. Messages which cannot be decoded, or for which submission fails for some other reason, are moved to an 'error' directory. The default behavior of the process is to ignore files in its input directory whose names start with "." or "~". Additional configuration allows only files with a particular suffix (e.g. "*.msg*") to be recognized, and can also enable high/medium/low priority sorting. On Unix platforms the server process simply scans its input directory regularly, looking for new files (the default is to check once per second). On Windows, use is made of filesystem notification functions, such that no regular scan is required.

Messages which cannot be submitted for some reason (e.g. failure to decode the contents of the file) are moved into an 'error' directory for manual analysis.

On Unix, p1fileserver should be run as a daemon, in the same way as the SMTP inbound channel. The following command line switches are supported:

- -c <channelname>: The channel as which the server process should run (this parameter is mandatory)
- -m: <mtaname>: (optional) The inbound MTA as which the server process should run.

On Windows the p1fileserver program should be installed as a Windows service. This can be done using the Services tab of the Switch Operations View in MConsole (or the standalone Isode Services Manager). Select the "Add" option, and configure fields as shown below:

Service Name	isode.pp.p1file
Description	Isode M-Switch P1 File Server
Executable path	*(SBINDIR)/p1fileserver.exe*
Service arguments	-c p1file

All other parameters can be left empty or with their default values.

Once the service is installed, you can start it using the Services tab of the Switch Operations View in MConsole (or the standalone Isode Services Manager).

If there is a requirement for multiple p1file servers to read from the same input folder (i.e. in a clustered MTA setup), the use of file locking needs to be enabled for the channel. This is done by setting the channel specific variable `use_lockfile` to the value `yes` or `true`.

4.2.10.15 P3 Service

The standard p3server channel operates as a process per connection: the `iaed` OSI listener process listens for an incoming connection on a specific Transport Selector from a P3 client application (i.e. a P3 User Agent or a P7 Message Store) and starts a new p3server process, passing it the connection. When the connection is terminated, the p3server process also terminates.

When large numbers of P3 client applications wish to connect to an MTA at the same time, this model has disadvantages, as a large number of processes will be started and connection establishment will be relatively slow and costly.

From R16.3 onwards, an additional P3 Service is available. This is a long-lived multi-threaded server process, which can handle multiple simultaneous connections. It implements all of the same functionality as the existing p3server channel, while providing better scalability.

This new server is not enabled by default. To use it, a number of configuration changes are needed.

- Modify the existing p3server channel configuration using MConsole. On the “Inbound” tab, delete the value in the “command” field, and change the Presentation Address so that it uses a non-default port. This is needed to prevent conflicts with the `iaed` OSI listener process.
- On Unix, rename */etc/isode/pp.rc.sample* to *pp.rc*, and edit it. Uncomment the “USE_P3” variable definition - this will cause the p3server process to be started as part of the MTA.
- On Windows, use the Isode Service Configuration GUI, or the Services tab in MConsole, to perform the “Install Isode Services” operation. Select the “M-Switch” service collection, and then ensure that the **Isode M-Switch P3 Server** (internally called isode.pp.p3server) service is checked on the name pane before pressing the Finish button.
- On Unix, stop and restart the MTA to run the new server. On Windows simply start the new service.

Any UAs will need to be reconfigured to use the modified Presentation Address on which the P3 server is listening. Otherwise no change is required.

The `maxinconn` channel-specific variable controls the maximum number of simultaneous connections the P3 Service will handle. This value is used as part of the service's initialization of its Queue Manager interface, requiring a restart of the P3 Service for any change to take effect.

4.3 Tables

This chapter describes the purpose and syntax of the various tables used by the message handling system to configure local users. As discussed earlier, the `aliases` table, the `users` table, the `domain` table, the `or` table and the `channel` table are the primary tables

used for routing and message delivery. There are also a set of tables containing authorization information, and a family of tables that provide mappings between X.400 OR-names and RFC 822 domain names. Specific channels may also have their own tables containing the information they require.

The General Syntax of Table Entries

In general, entries in the tables are in the form:

```
<key> ":"<value>
```

The only special character is : which, if it appears on the left hand side, must be preceded by a \ to escape it. All other characters are copied verbatim. However, some tables undergo further processing and may need other escape sequences. The following sections describe the purpose and details of the various tables.

4.3.1 Common

aliases auth.channel auth.mta auth.qmgr channel list channel users local shell list

4.3.1.1 The aliases table

The aliases table governs the handling of aliases for user names. It is used for several reasons. These include:

- Mapping non-users to user names. (e.g., postmaster to some userID)
- Redirecting users who have moved (e.g., fred to fred@foo.edu)
- Rewriting users addresses (e.g., jea to j.austen)
- Mapping aliases. (e.g., list-request to j.austen)
- Identifying ambiguous users

Entries in this table are constructed as follows:

```
<username> “:”<type> <value> <qualifier>
```

Where

username is the local name for which an alias is being created.

type describes the user name or address. This may be one of:

- `synonym`: a new address, this name replaces the original value.
- `alias`: a new address is given, but in some cases this name does not replace the original value. Aliases are not expanded for originators of messages whereas synonyms are. Aliases are also not expanded in the normalisation of addresses in message headers. Aliases, when expanded, add a redirect history element to the appropriate recipient of the message.
- `redirect`: this is similar to the alias type with the restriction that if the `recipient-reassignment-prohibited` submission extension is set in the envelope of the message, the expansion of the new address is not permitted and a delivery report is returned for that recipient.
- `ambiguous`: this indicates that the address is ambiguous and a delivery report should be sent. In this case the value, if present, is used as additional text for use in the supplementary information field.

The <value> is an address. It can be local or remote. The <qualifier> gives the interpretation of the value. By default this is assumed to be a local user. However, full addresses can be specified with the appropriate qualifier. These are:

- 822 : a remote RFC 822 user address is specified.
- X.400: a remote X.400 user address is specified.
- external: If this qualifier is present the name will not be looked up again in the aliases file. This can be used for complex mappings.

The qualifier and format is best illustrated by a sample extract of the table as shown below.

Example of Aliases Table

```
# sample aliases
#
mailgroup-request:alias Alice.Carroll
postmaster:alias Alice.Carroll
pp:alias postmaster
#
j.austen:synonym Jane.Austen
jane:synonym Jane.Austen
#
wth:alias wth@widget.co.uk 822
#
pc:alias “/I=P/S=Principle/O=Widget/PRMD=Widget Co/ADMD= /c=gb/” X.400
#
f.chopin:synonym f.chopin@foo.bar 822 external
```

The table includes an entry for a local alias `mailgroup-request` .

The value `Alice.Carroll` is used as a key for a further search in the `aliases` table, but will not replace the alias in the message header; if no match is found, and since the entry is of type alias with no qualifiers, the address is assumed to be a local one and is used for accessing the user table. Also, if a local submission arrives from `mailgroup-request` then checking and authorization will be done on the basis of `mailgroup-request` rather than `Alice.Carroll` .

The next entries indicate that `j.austen` and `jane` are both synonyms for `Jane.Austen` . Here, the user's address will be converted to `Jane.Austen` , and this value will be used in future table searches.

An entry is given for a remote RFC 822 user, `wth` . The value of

`wth@widget.co.uk`

is parsed as an RFC 822 address. Then follows an entry for a remote X.400 user, pc . The value

`“/I=P/S=Principle/O=Widget/PRMD=Widget Co/ADMD= /c=gb/”`

would be parsed as X.400 address. The quotes (“ “) are needed as the entry contains a space which is a field separator.

Finally, the entry `f.chopin` is an external synonym. The address `f.chopin` will be replaced by `f.chopin@foo.bar` for the originating address and further lookups of the aliases table will be disabled. It will not be replaced in recipient addresses to which delivery is being attempted. It may or may not be replaced for addresses in message headers depending on the tailoring.

Use of the utility `aliases` found in *tools/tables/aliases* may help you generate useful combinations of valid aliases from input data.

4.3.1.2 The users table

The users table determines the local delivery channel for a particular local user. It has the following format:

```
<username> ":" <channel> [ <mta> ] ["," <channel> [ <mta> ] …]
```

username can either be the local name of the user, or the wildcard value, "*", which will match any local user. "*" could be useful in several situations. For example:

- If most of the users for a local domain need to be routed to another MTA. The wildcard entry would come after the entries for those local users requiring different routing, and act as a catch all. This is likely to be the most common use of the wildcard entry.
- If address conversion is required for some users in the domain, and only some of the addresses that do not require conversion are known.
- Where the X.500 Directory is used to hold P3 delivery information for all local users, but routing is table based. A wildcard entry in the users table would avoid the need to add entries for all these users.

channel specifies the name of a local delivery channel on the MTA. This parameter is mandatory, but is ignored if a remote MTA is included.

mta is the local machine on which the mailbox resides. If no MTA is specified, the channel may be used for delivery to any MTA. If an MTA is present and is not the local machine (`loc_dom_mta`) then the message will be sent to that MTA by an appropriate channel and the channel parameter is effectively ignored - the normal routing tables will be used to reach that host. This should only be used for internal shuffling between several MTAs that are responsible for the same domain.

Example of Users Table

```
############## extract of 'users' table ##########################
Alina.DaCruz:822-local mta1.sales.widget.co.uk
Cristopher.Marlowe:822-local mta2.sales.widget.co.uk
Jane.Austen:822-local mta1.sales.widget.co.uk, p3 Peter.Principal:822-local m
X.400-users:list
info-server:shell
```

The example above illustrates how the various local users will receive mail, and on which machines their mailboxes reside. In this example `mta1.sales.widget.co.uk` is the name of the local MTA. Mail for Christopher Marlowe is to be sent to the remote MTA, `mta2.sales.widget.co.uk`. The appropriate channel table will define which channel should be used to reach this MTA. In this entry, therefore, the `822-local` channel entry is ignored.

A user can have several channels for local delivery. For example, `Jane.Austen` can have her RFC 822 mail delivered by the 822-local channel, and her X.400 format mail delivered by the `p3` channel.

An entry for a distribution list is also shown, and the entry for `info-server` shows the `shell` channel rather than a local delivery channel.

Using the wildcard entry for rerouting

```
############## extract of 'users' table ##################
Alina.DaCruz:p3 mta1.sales.widget.co.uk
Jane.Austen:slocal mta1.sales.widget.co.uk
Peter.Principal:p3 mta1.sales.widget.co.uk
```

```
info-server:shell
*:smtp mta2.sales.widget.co.uk
```

In the example above, `Alina DaCruz` and `Peter Principal` require their mail to be delivered by the `p3` channel. Mail for `Jane Austin` needs to be delivered using the `slocal` channel, and mail for `info-server` is routed to the `shell` channel. Mail for all other local users is routed to the remote MTA, `mta2.sales.widget.co.uk`, by the appropriate channel. Note that the `smtp` channel entry is therefore ignored, because `mta2.sales.widget.co.uk` is not the local MTA.

4.3.1.3 The 822 Local Table

The local table, held in the file `ch.local` , is an example of a table accessed by different channels; both 822-local and `slocal` channels use it in order to find out where and how to deliver mail to registered users.

The LHS is the registered users mail address. The format of the RHS is key=value. The following table of key/value pairs are allowed:

uid: The numeric userID used to deliver the message. This is set to zero on Windows.

gid: The numeric groupID used to deliver the message. This is set to zero on Windows.

username: A user name found in the password file. If this entry is set, it sets defaults for `uid`, `gid`, shell, home and directory.

directory: The directory to change to before starting delivery. This implies CWD for any files which use relative path names. If it is not set it defaults to home.

mailbox: The name of the mailbox if default delivery is being done.

shell: The user's shell and the one which is executed to run pipes etc.

home: The user's home directory.

mailformat: The default format to deliver mail in; this can be either pp for MMDF/PP style mailboxes, Unix for `sendmail` compatible style or `maildir` for `maildir` format delivery (the default is pp).

When using `maildir` format, the mailbox key must be specified as well, and identifies the root of the `maildir` directory structure into which to deliver.

Note: `maildir` format delivery is not supported on Windows

restricted: this is set to true if the user is restricted. In restricted mode the user cannot run arbitrary programs on delivery and cannot change the PATH environment variable.

mailfilter: The mailfilter file to use. This defaults to the name of the mailfilter global tailor variable in the users home directory. A value of none disables this feature.

sysmailfilter: The system default `mailfilter` file . A keyword of none′ disables this facility.

searchpath: The directory to look for binaries that the user can run in restricted mode. This variable defaults to *(BINDIR)* .

opts: Values in this field can include:

- options passed via $(opts) to mailfilter processing.

If the restricted mode is in force, then the .mailfilter file processing is reduced in functionality as follows:

- The user is not allowed to set the variable PATH to search other directories for programs. (More accurately, this variable can be set, it is just ignored.)
- The processes executed by the pipe command must be in the directory named in searchpath (defaults to *(BINDIR)*).
- The processes executed by the pipe command must be executable by the system call exec(2) . (For example; redirection and shell syntax will not be obeyed).

For an entry in this table to be valid, it must either contain at least a Unix login ID, or have a userID/groupID/directory to deliver to. For example:

```
Arthur.Katz:username=arthur home=/cs/users/vs1/arthur
Jane.Austen:username=jausten opt="poppasswd=kb5GD-yr18_K2"
Peter.Principle:username=pp
bug-filter:uid=32767 gid=1001 mailbox=/usr/spool/mail/bugs
mailformat=unix
```

NOTE: For compatibility with earlier releases the following format is allowed:

```
name ":" <unix ID> [<home directory> [<file>]]
```

If there is an entry for default in the table, this can be used to default any of the parameters.

4.3.1.4 The P3 Table

An example P3 table file is:

```
default: lpass="mtapassword", lmta="Widget.co.uk"
/I=F/S=Austen/O=Widget/ADMD=XX/C=GB/:rpass="x/XJeZt/EoZ5s",
   rpsap='"407"/Internet=193.63.86.1',
   rmta="cn=Frank Austen,o=Widget,c=gb"

/I=C/S=Bronte/O=Widget/ADMD=XX/C-GB/:rpass="j/HvctaySpDrg",
   rpsap='"407"/Internet=193.63.86.1',
   rmta="cn=Charlotte Bronte,o=Widget,c=gb" ||
```

The key to each entry is the OR-address of the MTS user. The fields used are:

lmta: The name of the MTA which will be provided in bind credentials to authenticate the MTA to the MTS user.

pass: The MTA password that will be provided in bind credentials to authenticate the MTA to the MTS user. The password is in plain text.

rpass: The MTS user password which must be provided in bind credentials to authenticate the MTS user to the MTA. If this field is absent, no credentials need be presented in a bind request or in a response to the MTS user. The password is encrypted. You can use *(EXECDIR)/tools/mkpasswd* to generate encrypted passwords.

rpsap: The Presentation Address which will be called by the MTS user when establishing an association to deliver messages. This is normally the address of a p7server process.

lpsap: The Presentation Address that the MTA will use as its calling address when establishing an association with the MTS user to deliver messages. This is an optional entry, and is rarely used.

rmta: The Distinguished Name (DN) of the MTS user, which will be used as the called *application-entity-title* when establishing an association with the MTS user. This is required when the MTS user is the Message Store.

4.3.1.5 The List Table

The `ch.list` table is used by the list channel to expand local distribution lists. Entries in this table have the format:

```
<listname> ":" [<moderator>   …] ","
file=<filename> <mail address>   … ","
description
```

listname: The name of the distribution list, e.g., `listname` .

NOTE: In order for the list to form a valid local address, the local part of the name, `listname` in the above examples, must have an entry in the local users table, specifying that messages addressed to the list should be delivered via the list channel.

moderator : The UNIX IDs/login names of the moderators of the list. This element of the table entry is purely for the use of the distribution list management tool, `mlist` . Anyone with one of the specified userIDs may use `mlist` to modify the contents of the list.

file: The file containing the members of the list. If *<filename>* is not a fully qualified pathname, the required file will be assumed to be under the `tbldir` directory structure. The format of such files is one member's address per line.

NOTE: The file must exist for `mlist` to be able to be used on the corresponding list (i.e. `mlist` is unable to create the files only modify them). Any lines in this file starting with the character # are ignored.

mail address: An address of a member of the list.

NOTE: Because they are specified in the list table itself, `mlist` cannot be used to modify such members.

description: A short description of the purpose of the list. This description is output by `mlist` when requested.

NOTE: In order to form a valid entry in the list table, each entry must be on one line and must contain two and only two commas.

Comment line in tables start with a # . The list table extends this mechanism further. Comment lines that are printed by `mlist` start with #Comment: . This allows the list table to have two levels of comments, one for the editor of the list table and one for the user of `mlist` .

Example of List Table

```
# extract of 'list' table
# Comment: -*- Example lists -*-
filelist:jea, file=/crg/pp/mainlist, a list with members in a file
strlist:pp|jea, j.austen|p.principle|a.mole, a list with members in a line
mixedlist:aam, a.mole|file=mixedlist, another list
```

If you have a large `sendmail` aliases file to convert, you may find the utility `make-lists(8)` of use. `make-lists(8)` is found in the *Tools* directory. This utility attempts to convert lists of names into a format suitable for use in the `list` channel.

NOTE: It does not convert all aliases table entries, only lists.

4.3.1.6 The Shell Table

The `ch.shell` table is referenced by the shell channel when converting from the recipient address to the commands to execute in its place. Entries in this table have the format:

```
<address> “:” <user id> “,” [<timeout period>[‘|’<qualifier>]]
“,” <command line to execute>
```

address: specifies the recipient address, e.g., shellprog .

NOTE: In order for this to form a valid local address, the local part, shellprog in the examples, must have an entry in the local users table, specifying that messages sent to the address should be delivered via the shell channel. This address can be just the local part of the address, the whole address, or the special key “*”, which is used if no specific entry for the address is found. The addresses which are to be delivered to the particular shell channel, should be configured to route to this channel. This is done for Directory based routing by setting the MTA channel appropriately for the user in EMMA, or for table based routing by setting the channel in the corresponding user's table entry.

user id: specifies the user to run as when executing the commands. This ID is resolved using */etc/passwd* . Alternatively it may be specified as `<uid>/<gid>` to run as an arbitrary user and group.

timeout period: specifies how long, in seconds, the commands should be allowed to run before the `shell` channel kills them off. If the timeout period is not specified, the commands are allowed to run for the default period of time, five minutes. If the period is zero then the shell channel will not enforce a time limit for the commands.

qualifier: qualifies the behaviour of the `shell` channel. This qualifier is optional appended to the timeout period with a "|" separating the two values. If the qualifier is set to the string solo then the shell channel will restart (fork) the command line for each bodypart.

command line: specifies the command line to execute. If the program name is not a fully qualified pathname, the program is assumed to be under the directory specified by `chandir`. Arguments of the form `$(key)` will be expanded as described below.

Shell Channel Expansion Macros

Key	Expansion
sender	the address of the sender of the message (original format)
senderdn	the distinguished name of the sender
822sender	the address of the sender (rfc822 format)
400sender	the address of the sender (X.400 format)
recip	the address of the recipient of the message (original format
recipdn	the distinguished name of the recipient
822recip	the address of the recipient (rfc822 format)
400recip	the address of the recipient (X.400 format)
qid	the queue ID of the message (e.g. msg.12345–0001)
ua-id	the user-agent ID of the message
p1-id	the P1 ID of the message
userid	the user name of ID of the delivery process
groupid	the group name or ID of the delivery process
size	the size in bytes of the message
channelname	the name of the delivery channel

When running, the `shell` channel pipes all the message bodyparts, in order, to the program's standard input. The `shell` channel understands an exit code of 0 as success and anything else as failure. If failure is reported and output from the process is detected, this text will

be used for the supplementary information of the delivery report. Additionally, if the message starts with the format <NUMBER>/<NUMBER>: then this will be interpreted as an X.400 reason and diagnostic code.

An example of a `shell` table is shown below.

```
################## extract of ‘shell’ table ##############
nullshell:pac,60, sh -c “/bin/cat > /dev/null”
useful shell: jpo, , usefulshellprog $ (ua_id)
lynx:pp, 0|solo, /usr/ucb/lpr -Plynx
```

Chapter 5 Content Checking

This chapter covers the advanced Content Checking features of the Isode M-Switch, giving detailed descriptions of the Quarantine system and how it fits into the Messaging Audit Database

5.1 Advanced Message Audit Database Features

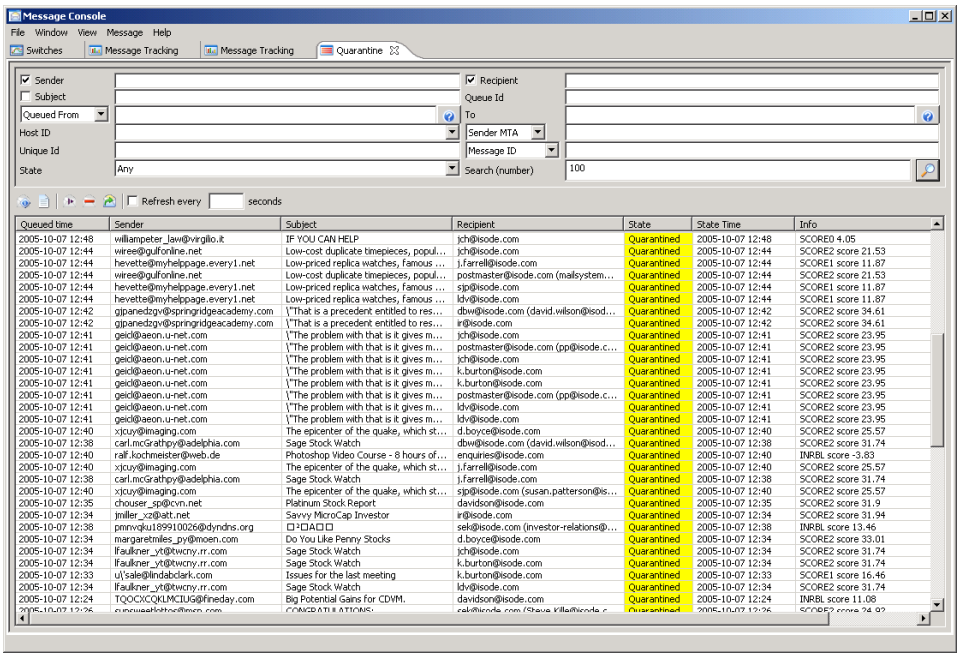
5.1.1 Quarantine

If you are using the content checking features of the M-Switch described in M-Switch Administration Guide you can enable the M-Switch Quarantine. This feature enables messages which fail the content checking checks to be placed into quarantine. For more details on the Message Audit Database (on which the quarantine system depends) see M-Switch Administration Guide

Messages in quarantine can be released and resubmitted into the MTA. M-Switch contains features to allow this by either the recipient or an operator.

An operator wishing to do this must search for the message using the message tracking feature in the quarantine view shown in Figure 5.1, “Quarantine view”.

Figure 5.1. Quarantine view



Right clicking on the message brings up a similar set of options to the message tracking view described in M-Switch Administration Guide with the addition of an option to release the message from quarantine.

5.1.2 Acknowledgement tracking

Messages traversing the MTS can result in delivery reports (positive or negative) and /or read receipts (also positive or negative).

The collective term for delivery reports and read receipts is 'Acknowledgement'.

Using the **Acknowledgement Tracking** view you can perform correlation of acknowledgements with subject messages. This allows you to check if messages for which positive delivery reports have been requested have received a delivery report and/or read receipt.

The **View** enables you to search for and display messages based on various combinations of requests and actual acknowledgements.

Use the **Expect ... within** and **Times start when** settings to specify how much time is allowed to pass before a missing acknowledgement will be treated as indicative of a possible problem. These times can vary according to message priority.

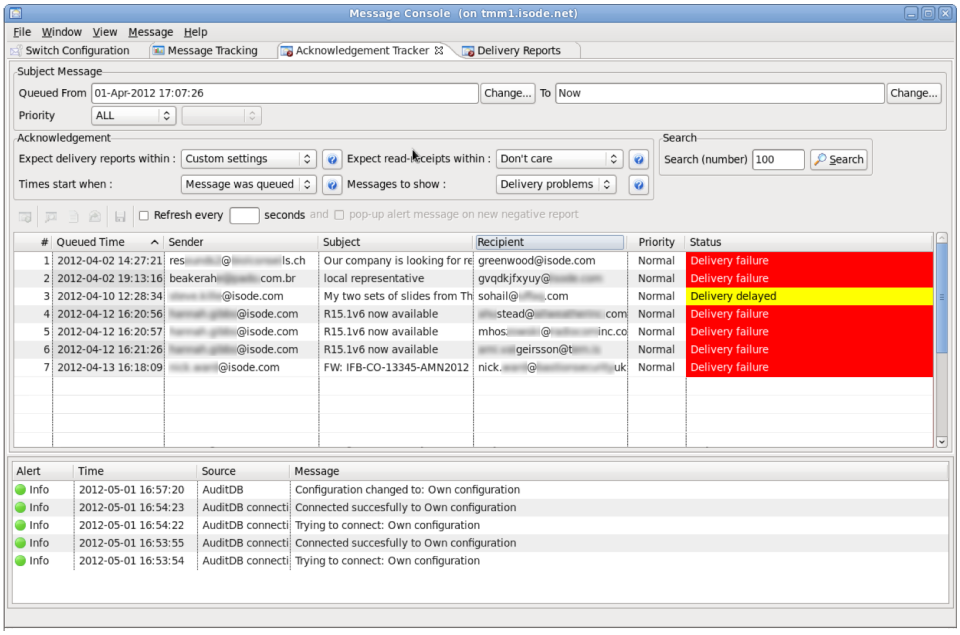
Use the **Messages to show** option to control which messages are shown, based on the state of their acknowledgement. For example, you can choose to show only messages which have had successful delivery reports, or all messages which have not yet had an acknowledgement. Various "preset" values are offered, or you can use the **Advanced...** option to generate a filter which is specific to your requirements.

A typical use of this view is to display all messages for which an acknowledgement is expected but has not been seen, so that you can take actions such as redirecting or resubmitting them. See Figure 5.2, “Message acknowledgement view” for an example of how such a display would look.

You can set a certain threshold to configure the time in which the reports or read receipts are expected. This time can vary according to the priority of the message.

You can select that messages in particular states such as successfully delivered/read; delivery failure; etc are displayed.

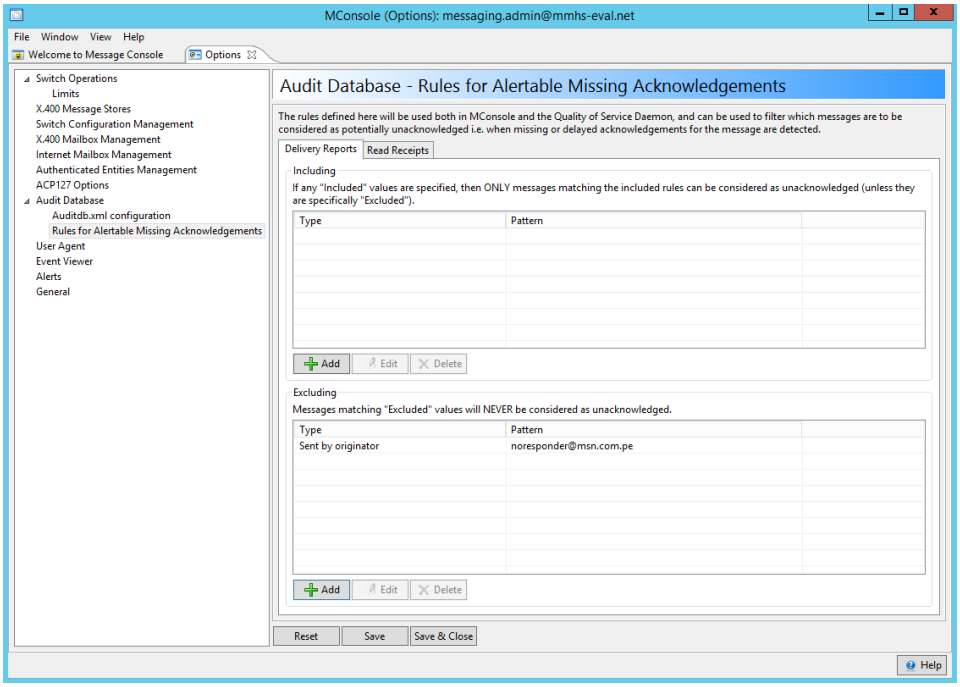
Figure 5.2. Message acknowledgement view



It may be that you know that there are certain addresses which will never generate acknowledgements (for example if they are addresses outside your organization which do not reliably generate delivery reports).

You can configure the Audit DB as shown in M-Switch Administration Guide to set rules by which messages are selected as being displayed as having missing alerts. You can do this by specifying addresses which will never be considered as being unacknowledged, or by specifying messages which will be considered as unacknowledged (unless the message also matches an exclude).

Figure 5.3. Message acknowledgement config



5.2 Content Checking

5.2.1 CCCP Checking Channel

The Content Checking and Conversion Protocol (CCCP) is a means for a checking channel to communicate with a server process which can perform both content checking, and also change the content. It can cause the message to be non-delivered, or deleted for each recipient, or the message can be redirected for a recipient.

This checking channel can be used in the place of the standard checking channel. They cannot be used together.

Which channel is selected for checking is configured in exactly the same way as for the standard checking channel, using information in the `auth.channel`, `auth.mta` and `auth.user` tables, and selecting the channel using the `content in` values, and also using the `recipient` address looked up in the channel's mtatable (if any).

5.2.1.1 CCCP Channel tailoring

The channel type should be '`check`'. The program should be `cccpchan`. The Content in should specify the content types which this channel checks. The channel may have an mtatable, which is used to specify the recipient addresses for messages to be checked by this channel.

Currently, a channel specific variable is used to configure the connection information for the CCCP server. The variable is called `server` and the value can be `hostname[:port]`

for a TCP connection to that host. The hostname can be an IP address, and the port number can be omitted, if the default port of 18003 is to be used. On Unix systems, the connection can also be made to a Unix socket, which is specified by a value which is the full pathname of the socket (i.e. it must start with '/').

5.2.1.2 Checker Channel Configuration

The standard checking channel has a configuration file which controls how the channel processes different components of the content being checked, and how changes to that component can be made, if required as a result of the action to be taken on checking. The file is an XML file which is very similar to that used by content conversion. It is located in the same way, except that the default name is *channel-name*-`checker.xml`.

Rather than output information for different content types, the checking channel configuration has check nodes, which are very similar to `convert` nodes for the content conversion. Each check node matches content components using the same features as `convert` nodes. The node has filters, which are used to extract and convert data, and also perform specific checks. In addition, a check node can have `keeper`, `repairer` and `replacer` nodes. These have the same format as shaper channel convert nodes. They are used to make changes to the component for the resulting content, for the cases where the component is to be retained, but changed in some way. `keeper` is used when the old version is to be annotated, `repairer` is used when the enclosed data is to be changed for repaired data, and `replacer` is used to generate a new component from replacement text. If the component is to be retained, and there is no keeper node, then there is no change. There are some specific filters provided for this, and standard shaper channel filters are also used.

5.2.1.3 Checker Channel Filters

:bodyscan

This is used for anti-virus scanning. The canonical data of the component is extracted and passed to the anti-virus software. It takes no parameters.

:textscan

This is used for text regular expression matching in the checking channel. It should be passed text converted to UTF-8, if the canonical data is not already UTF-8. It takes no parameters.

:wordscan

This is used to find words which are checked against the word scores. The input data should be UTF-8. It takes no parameters.

:htmlscan

This applies the HTML regular expressions. The input should be UTF-8, and the output is the data without the HTML markup (e.g. suitable for passing to `:wordscan`). It takes no parameters.

:mimeheaderscan

This scans the heading of a MIME body, including multiparts and normal bodies. It takes no parameters.

:rfcheaderscan

This scans an Internet message heading, looking at fields such as the From:, To: and Subject:. it takes one parameter `outer` which if present indicates to the checking process that this is the outer message heading.

:ipmheaderscan

This performs limited scanning of an X.400 IPM heading, e.g. of the subject. It takes no parameters.

:uudecode

This extracts uuencoded from a text body, and generates a new component from that data, for checking.

:bodyrepair

This provides to later filters the canonical data as repaired by the anti-virus software. It takes no parameters.

:bodyreplace

This provides to later filters the text to be used to replace the body part. Other filters should be used to create the correct body part type for the enclosing content type. It takes one optional parameter, text. If present, this is the replacement text. Otherwise the replacement text from the channel rule is used.

:rfcheadernotate

This annotates an Internet message heading, using information provided by the action rule. The subject can be annotated, and additional heading fields added. It takes no parameters.

5.2.2 Virus Checking for AMHS Installations

A common requirement for AMHS installations is that File Transfer bodyparts within X.400 messages are checked for viruses, while other bodypart types (e.g. IA5Text) are not checked (to improve performance).

A number of configuration steps are needed to set this up:

- Create and configure a suitable checker channel. This will normally be called `mhscheck` if it is intended for X.400 content checking. It will have `content in` values of `p2,p22`. You will need to ensure that the channel's `Anti-Virus` tab has settings for the Anti-Virus engine which you are using. A suitable Shaper configuration file which causes only File Transfer Bodyparts to be checked is included in the release (*(SHAREDIR)/switch/mhscheck-checker-ftbp-only.xml*). This must be selected as the `XML configuration file` option for the channel.
- Create a directory called *(ETCDIR)/switch/msgcheck*, and create a file named *standard.rule* in this directory. The file should contain the lines:

```
rule program VIRUS {pushv virus_notok pushv virus_replaced add}
rule action VIRUS reject
rule priority VIRUS 100
rule status VIRUS 5.8.48
```

  The second rule configures the action to be taken if a virus is detected; `reject` indicates that a non-delivery report should be generated.

  The final line configures the X.400 non-delivery report reason and diagnostic codes which will be used in any non-delivery report which is generated. The string `5.8.48` indicates that a reason code of "transfer-failure-security-reason" and diagnostic of "unable-to-complete-transfer" will be used.

- Optionally configure supplementary information. The supplementary information for non-delivery reports is generated in the Tcl script which provides the interface between the checker channel and the anti-virus package. For example, if you are using the ClamAV package, the Tcl script is *(LIBDIR)/msgcheck/clamav.tcl*. By default the string used is

simply `Found $virusname` (where `$virusname` is substituted with the virus name string returned by the anti-virus package. If necessary, this string can be modified; no further action is needed.

- Run the *cachebuild* program to build the `mhscheck` channel's configuration cache. The program takes a single argument which is the name of the channel for which it is to build the cache.
- Set up an authorization rule which causes messages to be checked. This can be as simple as a rule of type `check` with the `mhscheck` channel selected: this causes all messages to be checked. If required, filters can be used to restrict checking to messages which arrive on specific inbound channels or from selected Peer MTAs.
- Configure your anti-virus package so that it can be used by M-Switch.
- Test the configuration using the standard EICAR sample virus.

Chapter 6 Content Conversion

M-Switch can be configured to modify the contents of messages. This includes the ability to act as a Gateway between the threee principal protocols of X.400, Internet and ACP127 messages.

6.1 Content Conversion in M-Switch

Purpose of Content Conversion

Normally, M-Switch passes the message content unchanged and it does not look inside the message. However, sometimes it is necessary to make changes to the content of a message.

- Converting between Internet and X.400 contents in a MIXER gateway.
- Changing addresses within the message heading in a boundary MTA.
- Conversion of one body part type to another type which the recipient will accept.
- Verification of S/MIME signed messages and extraction of the signed body.
- Using S/MIME to sign messages.
- Extraction and insertion of security labels.

Content conversion can also move information from the message envelope into the content, and move information from the content into the envelope.

- When Content Conversion is done
- General Configuration for Content Conversion
- Shaper Channel Operation
- Shaper Channel Configuration
- Specific Conversion Filters
- MIXER conversion features
- Security Configuration
- DKIM Configuration

6.1.1 When Content Conversion is done

The routing calculation (see M-Switch Administration Guide) results for each recipient of the message a set of channel and mta pairs. The message has an inbound content type, an inbound header type (except for some X.400 content types) and, for X.400 messages, a set of "encoded information types". These are compared with the content types, header types and encoded information types which are accepted by the channel, MTA and recipient. If the message can be accepted for the route by the recipient, then content conversion is not performed, and the message is passed unchanged. If there is no route which can be taken by the unchanged content, then content conversion needs to be performed.

M-Switch then looks for a suitable conversion channel. This is chosen from the channels which have the type `shaper`, and for which the inbound content type of the message has a match in the **Content In** value for the channel. If no suitable shaper channel is found, then the message is rejected or non-delivered.

A message with more than one recipient may require conversion for some recipients and no conversion for others. If conversion is required for more than one recipient, then the conversion can be different for different recipients. This is handled by a single channel.

6.1.2 General Configuration for Content Conversion

In order that Content Conversion be performed when required, it is important that various aspects of the overall M-Switch configuration be considered.

Inbound Channel Configuration

Sometimes it is desired to perform different conversions on the same basic content type depending upon the source of the message. For instance, messages which are newly submitted might have various updates performed such as inserting a missing Message-ID.

This is achieved by separating the sources using different inbound channels. Then the channels can be configured with a content subtype. This subtype can be used in the selection of component converters.

Outbound Channel Configuration

When the inbound message is compared with the outbound route, the outbound channel contributes some items.

Content out
: This is set to the list of the content types which this channel will accept. If the list is empty, however, the channel will accept any content type.

Content subtype out
: This is an optional value which enables different conversions to be performed for the same basic content type.

Bodyparts out
: This is set to the list of encoded information types which the channel will accept. If the list is empty, then any encoded information type is accepted.

Component converters can specify the outbound subtype required, and the encoded information types they produce. These data are used to select the converters to be used.

Recipient Configuration

For X.400 recipients, it is possible to configure in their Directory white pages entry attributes indicating:

- the content types accepted by the recipient
- the encoded information types which are permitted or forbidden

6.2 Shaper Channel

6.2.1 Shaper Channel Operation

First, the channel will 'explode' the content to discover the components of the message. That is the different headings and body part types within the content. For MHS content, it is possible to suppress this, and the message content is considered as a whole, with no subdivisions. The message envelope is also considered as a component of the message, as the output heading of the converted content can contain some fields from the envelope.

It is possible for the channel to check S/MIME signatures at this stage. If a signed body contains an encapsulated S/MIME body, then the exploding process will consider that body, which could itself be a multipart containing other bodies. Also, if the S/MIME signature contains a security label, this is made available for use within the message envelope (if an Internet S/MIME message is being converted to X.400, for example).

Each recipient has a set of pairs of channel and MTA to which the message can be transferred. The shaper channel considers each recipient separately, and considers each outbound possibility. However, if the message has more than one recipient, and two or more recipients need the same conversions, then the recipients can be associated with the same output content from the conversion process.

Each output route for each recipient is considered in turn. If the route has multiple content types, then each content type is considered, along with the encoded information types allowed or excluded for the channel and recipient. The shaper channel has configured a number of possible output content types, which gives the conversions for different components in such a content. For each component in the content to be converted, the appropriate conversion is identified by matching the properties of the input component. Also, if the conversion has a set of encoded information types, these are considered to check if the conversion can be used. For some components there are defaults conversions. Matching conversions to components can use feature of the parameters of the component, such as the body part type. Also, there is a limited ability to match the data within the body (c.f. the file command found in Unix systems).

It can be that more than one conversion matches the input component. In this case the lowest cost conversion is used, the cost being a parameter of the conversion. Therefore, if configuring conversions, more general matches should have a higher cost than more particular matches. Also, less preferable conversions (e.g. because there is data loss) should have higher cost that more preferable conversions.

It may be that there is no content type accepted by the channel, or no set of conversions which can be used on the input content to convert it to the proposed output content type. In that case, the message is non-delivered for that recipient. It is also possible that the originator of the message may have indicated that the message cannot be converted, or that conversion with loss is prohibited. Then the message is also non-delivered. A conversion can be flagged as lossy for the latter case.

If conversion is possible to the target content type, each conversion to be used has a cost, which gives an overall cost to the conversion. If there are different possible output content types, then the output type used is the one with the lowest overall conversion cost.

Having chosen the content type and set of conversions, the shaper channel generates the output content by using the appropriate ‘flattener’. This considers each input component, and applies to it the conversion action. This will often involve taking the information for the input component, and applying a sequence of filters.

6.2.2 Shaper Channel Configuration

6.2.2.1 Tailoring Configuration

Channel Type
should be `shaper`

Program
should be `shaper`

Configuration file
If omitted, the name of the file is derived from the name of the channel, by adding `-shaper.xml`. You can configure a specific file by giving its name or its full pathname.

If the configuration file name is not a full pathname, then the program looks in two places:

1. subdirectory switch of the *ETCDIR*

2. subdirectory switch of the *SHAREDIR*

So, on Windows it will look, typically, first in `C:\Isode\etc\switch` and then in `C:\Program Files\Isode\share\switch`. For Unix it will look first in `/etc/isode/switch` and then `/opt/isode/share/switch`

Default configuration files for single protocol and MIXER configurations are provided in `SHAREDIR/switch`. If you want a default MIXER setup, you should set the configuration files as follows

mimeshaper
`mimemixer-shaper.xml`

mhsshaper
`mhsmixer-shaper.xml`

If you need to change the configuration file, you are advised to place your edited file in ETCDIR/switch. The SHAREDIR/switch location is for files provided by Isode.

6.2.2.2 Tailoring for S/MIME and Security

The channel can verify S/MIME signed messages, and can generate S/MIME signed messages. It can also process security labels from S/MIME messages, the envelope of X.400 messages, and through heading and first line of text (FLOT) text labels.

Currently the configuration for these areas is performed by setting channel specific variables (in the Advanced tab in EMMA). There are two methods for setting up data for S/MIME. From R15.2 a database is used. This can be a permanent file, in which case the configuration uses:

securityDB
File path to security database. If relative, then relative to *ETCDIR*.

securityDBpass
Passphrase for the security database, if required. The value can be encrypted with the M-Switch server master key.

signing-id
The URI identifying the private key, or corresponding certificate, used to sign messages.

The use of a permanent database is required for encryption and decryption. For encryption, the database should have loaded suitable certificates to be used for key encipherment or key wrapping. The encryption code will look for a certificate using a suitable `subject alt. name` value. For a recipient this will be either an rfc822 address or X400 address. For a peer MTA, this will be the DN of the peer's channel configuration entry, or a dnsname of its hostname. It may be necessary to associate a meta-data value with the certificate in the database, so that the shaper channel can identify the correct certificate to use when encrypting for that user or peer MTA.

The older channel variables can be used to configure an ephemeral database for signing and verification:

x509_p12
The pathname for a PKCS#12 file which contains the private key and certificate for the digital identity to be used for signing messages.

x509_pphr
A text file containing the passphrase for the PKCS#12 file.

x509_certs
The pathname of a directory which contains certificates which are used as trusted certificates when verifying signed messages. In addition, if a PKCS#12 file is specified, any trust anchors within that are used.

x509_crlcheck
Should contain the value true or yes to cause the certificate verification procedure to attempt to obtain certificate revocation lists. LDAP access for this uses the MTA-wide LDAP access information.

The shaper channel uses configuration files described in M-Switch Administration Guide) for label extraction and insertion. These are described in the following sections. The channel variables can be used to set up the security policy called `@default@`.

security_policy
The pathname of the security policy file to be used. This should be in XML format

default_label_file
The pathname of a file containing a label in XML format. The label is used when generating an S/MIME signed message and there is no other label available.

default_label_policy_id
Object Identifier in numeric *oid* format which is used when an X.411 label has no policy ID. This is only used if there is no security policy.

6.2.2.3 Configuration File

The configuration file is an XML file providing information to the channel program about the desired changes to content. The root node of the XML is '`shaper`', and this defines the '`exploder`' for the input content. This is the action which understands the structure of the content of the message, and so can deal with the components of this content. It is important that the exploder matches the content type presented to the channel. So the channel's content in setting should match the exploder setting. Internet content should use `ppmime` and X.400 content `ppmhs`.

At the next level there can be `param` nodes, which alter the action of the exploder. There are output nodes, which have a content type and a flattener. The flattener knows how to put a message back together from its components.

Each output node has a set of convert nodes. Each of these describes how to handle the conversion of a component of the input content into a component of the output content. The converter can have some parameters, using `param` node. It can have match sub-nodes, which specify restrictions on the properties of the input component which can be processed by the converter, and data sub-nodes which can give restrictions based on matching the data within the component. it can have a set of eit sub-nodes, which specify the encoded information types associated with the component resulting from the conversion (X.400 output content only). The eit values are used to ensure that the resulting content is acceptable to the channel and recipient. The converter can have a sequence of filter sub-nodes, where each filter performs some change on the data for the component.

6.2.2.4 Exploder configuration

There are some parameters which control the exploders.

ppmime exploder

smime-body
A space separated list of MIME body sub-types which are recognized as S/MIME bodies. This list is checked if the MIME type is `application`. Typically this is set to "`pkcs7-mime x-pkcs7-mime`".

smime-sig
A space separated list of MIME body sub-types which are recognized as S/MIME signatures, for the preceding MIME body. The list is checked if the MIME type is `application`. Typically, this is set to "`pkcs7-signature x-pkcs7-signature`".

sime-verify-fail
The value gives the action to take on a failure to verify a signature within an S/MIME message. The values resulting in action can be `abort`, which will cause the message

to be non-delivered, or `mark` which will add a heading to the body indicating the failure. Any other value, or not specifying this parameter will allow signature failures to pass. However, such a failure is always logged in the audit log.

ess-label-priority
The label priority associated with ESS labels in signed messages.

sio-label-priority
The label priority associated with labels found in SIO-Label header fields.

ppmhs exploder

This has parameters based on content types. If there is a parameter for the specific content type, then that is used. If there is none, and there is the parameter named any, then that value is used. The value is a list of keywords for that content type.

noexplode
Do not explode the content into its components. The result is that only “whole content” conversion can be done. For instance, conversion to the “X400 wrap” MIME body.

pct
The content is a Protected Content Type, i.e. an X.400 content using Cryptographic Message Syntax (CMS) for a signed or an encrypted content. This action makes the type of the protected content available to the conversion process.

ess-label-priority
The label priority associated with ESS labels in signed messages.

6.2.2.5 Output node configuration

The output node has attributes:

type
specifying the output content type. There is a special value p2orp22 which is used if p22 is in the list of acceptable content types. However, if P772 and/or P2 also appear in that list, then the actual outbound content type may not be P22:

- If P772 is in the list and the message contains any P772 heading extensions, then the content type is set to P772
- If P2 is in the list and the message contains no elements which are not permitted in P2 (e.g. heading extensions), then the content type is set to P2.
- Otherwise the content type is set to P22.

flattener
Specifies the flattener to use. This must match the resulting output type. Use ppmime to generate Internet content, and ppmhs for X.400 content.

6.2.2.6 Converter Configuration

The main definition can have the following attributes:

type
Indicates the type of component to which the converter applies

cost
A positive integer indicating the cost

action
Indicates the action to be taken by the converter

outtype
Indicates the content sub-type output by this converter

The different types of component are:

body
A standard data body

header
A heading

message
A message body, i.e. which contains heading and bodies

multipart
A message body containing body parts

envelope
The message envelope

content
The overall message content

The actions which can be taken are:

keep
No change

discard
Discard this component, this implies information loss.

convert
Perform conversion using the filters

implicit
Perform conversion using the filters, but forbidden if implicit conversion is prohibited

lossy
Perform conversion using the filters, but the conversion loses information, so forbidden if conversion with loss is prohibited.

ignore
Take no action, this does not imply information loss.

merge
Combine the component with the previous component

wrap
Combine the enclosed components into a component of the appropriate type

content
The converted component becomes the whole output content. This is used to convert a MIME "X.400 Wrap" body into an X.400 content.

extract
Used for signed items (`multipart/signed` or `application/pkcs7-mime`) and replaces the body with the signed body, without the signature.

insert
The component and its children are flattened, and the result is then fed into the filters for the converter.

sign
Generate an S/MIME signed body. If applied to a heading, then the following body is signed. If applied to a content, then the signed body is a message/rfc822 containing that content. This action has converter parameters which control the signing.

encrypt
Generate an S/MIME Enveloped Body (i.e. encrypted). Applied to a heading to encrypt the following body, to a content to generate a message/rfc822 body which is then encrypted. For MIME (822) output only. This action has converter parameters which control the signing.

triple-wrap
Generate a Triple-wrapped content, i.e. a signed content within an encrypted content within a signed content. Applies only to a content component, and for X.400 output only. This action has converter parameters which control the encryption.

failed
Indicate that the conversion has failed

If there is no matching converter for a component, then defaults apply. For a body or header, the default is the equivalent of a ‘failed’ converter. For a message, multipart or content, the default is a “wrap” converter, and for an envelope, an “ignore” converter.

Signing configuration

If the action for a converter is sign, then this can be controlled by a parameter named `SmimeAction`. If this is present has the value `signed`, then the signing will generate an `application/pkcs7-mime` body part. This embeds the signed body within the data of the S/MIME body. If the parameter is not present, or is some other value, then the signed body is combined with an `application/pkcs7-signature` in a multipart/signed body. Here, the signature information is separate from the signed body.

When generating an X.400 signed content, it is necessary to specify the content type to be used when generating the content to be signed. This is done using the `input-type` parameter. The default value is "`p2orp22`".

Encryption configuration

If the action for a converter is encrypt or triple, then there are some parameters for the converter which control the encryption process.

algorithm
The name of the algorithm to use as the content encryption algorithm. This uses the OpenSSL algorithm names. If omitted, "`aes-128-cbc`" is used.

encrypt-for-peer
If set (to any value), then a suitable public key is sought for the relevant peer-MTAs as well as the recipients of the message.

verify
If set to "`true`", the recipient or peer MTA certificates are subject to certificate verification.

When generating an X.400 triple-wrapped content, it is necessary to specify the content type to be used when generating the content to be signed. This is done using the input-type parameter. The default value is “p2orp22”.

Match configuration

Match items are used to match attributes of the component being considered. Each match item must match for the converter to be considered. Each match has the name of a component attribute, and an optional value. If the value is omitted, then the match is "`true`" if the attribute is present in the component, and "`false`" otherwise. If the value is present then it is a regular expression, and the result of the match depends on the match of the regular expression on the actual value of the component's attribute.

The match element can have an attribute `@child@`, which enables the test to be on attributes which belong to a component which is a descendant of the component for which the match is being evaluated. The first child of the component is always considered. The value of the attribute is an integer giving the number of levels to descend. This tests an attribute which is the immediate child:

```
<match name="Type" child="1">message</match>
```

The attributes available depend upon the exploder, which creates the components.

Data match items are a method of checking the data within a normal body part. They check the ‘canonical’ data for the body, which is the data with any transfer encoding removed. A data item has attributes:

type
The data type to be matched

offset
The offset of the value within the canonical data

Data types can be:

byte
A single byte

short
A two byte integer in little-endian order

long
A four byte integer in little-endian order

beshort
A two byte integer in big-endian order

belong
A four byte integer in big-endian order

string
A string value.

Integer values for the offset and the value to be matched can be expressed as decimal, octal (leading ‘0’) or hexadecimal (leading ‘0x’).

String values, if they end in “`/c`” or “`/C`” then the characters before the ‘`are matched without regard to case. If there is no`‘ in the match value, then the string can contain escapes for non-printing characters. These are similar to normal ‘C’ string escapes.

For a converter generating an X.400 body-part, the *encoded information types* which are associated with the body-part are specified by eit nodes. The value is either the standard X.400 name for a basic encoded information type (c.f. X.411) or is the object identifier expressed in numeric oid format for extended encoded information types.

Filter configuration

A filter has a command attribute which identifies the filter. The name comprises the name of the library containing the filter, a colon, and then the name of the filter within the library.

A filter may have configuration parameters.

6.2.3 Specific Conversion Filters

6.2.3.1 Internet Message filters

ppmime:header

This filter performs various actions on an Internet message heading, including the heading within a MIME message/rfc822 body.

The filter can perform *address normalization*, which means following address synonyms, and ensuring that domains are fully qualified. Standard heading fields which contain addresses are considered, and other fields can be added using a parameter with the name `address` and the parameter value the field name.

The parameters `external` and `internal` control which address synonyms are followed. This enables one to convert between internal and external versions of an address. These are normally used in conjunction with suitable header sub-types. If neither of these is present, then no address normalization is performed.

If the parameter `normalize-all` is present, if an address has a route, then all domains within the route are normalized. Otherwise only the leading domain is normalized.

If the parameters `add-date` or `add-message-id` are present, then if the heading does not have a Date field or a Message-ID field, respectively, then a suitable field is added. These should only be used when the message has been locally submitted.

The parameter `strip-trace`, if present, causes 'trace' fields (e.g. Received fields) to be removed from the heading. This can be used to suppress information about internal networks at a boundary.

The parameter `strip-most`, if present, causes most fields to be removed, apart from a list of standard fields. This list can be added to by using the keep parameter. Each such parameter defines another field which is to be retained.

The parameter fold controls the folding of field values. The value should be an integer between 20 and 200, which defines the target length of a line. Folding white space is inserted to keep heading lines within that line length, if possible.

[R15.0 and later] The parameter `add-disposition-notification-to` forces the addition of a `Disposition-Notification-To` field, if one is not already present. If the parameter name is not one of the above, then it is treated as a heading field name. The action to be taken depends upon the value of the parameter:

address
: The field contains addresses, which will be normalized.

keep
: Prevent the field from being removed when strip-most is used.

strip
: [R15.0 and later] Strip the field from the heading.

ppmime::charset2utf8

This filter is used to convert the data in a MIME text body from the character set to UTF-8. Most body part generation filters require text in UTF-8 as input, so this is a prerequisite for this.

If the parameter unknown is set, then if the character set for the body is not known, then the character set which is the value of the parameter is assumed. If this parameter is not set, and an unknown character set is found, then the conversion will fail.

ppmime:encode

This filter is used to generate the MIME encoded data from the canonical data for a body. It has one mandatory parameter, Encoding, which should have a value which is a valid value for a MIME content transfer encoding (c.f. RFC 2045).

ppmime:text

This filter is used to generate the canonical data for a MIME text body. If the parameter `charset` is specified, then this gives the output character set, otherwise us-ascii is used.

The parameter `StringFirstPageBreak` can be specified with the value true to remove any initial page break (the character sequence CR FF).

ppmime:insertsmimeauth

This filter is used to insert a heading field `Authentication-Results` recording the result of checking signatures on S/MIME signed data.

The optional parameter `authority` can be set, its value giving the authority field in the field value. If this is not set, then the value of the MTA parameter `loc_dom_site` is used.

6.2.3.2 X.400 Filters

ppmhs:downgrade

Some X.400 IPM body-parts come in equivalent forms, one the 'basic' form and one using the extended body part form. Only the former are permitted in X.400 1984 IPMs (P2). This filter converts the extended form to the basic form, so that the message can be downgraded to P2. It has no parameters.

ppmhs:heading-downgrade

This filter removes any features from a P22 heading which are not permitted in a P2 heading. It has one parameter, downgrade, which controls how OR-addresses are downgraded. It takes these values:

x419
: Follow the downgrading rules as specified in X.419

common
: In addition to the X.419 rules, a printableString CommonName attribute is converted into a DDA with the type 'common', and value from the attribute.

rfc1328
: Use downgrading as specified in RFC 1328.

ppmhs:general-text

Take the canonical text input (in UTF-8), and generate a general-text body part using the configured character sets. The character sets are configured using

Charsets
: The value is a space or comma separated sequence of integers. Each integer is the ISO registration number of a character set.

Charset
: The value is a mnemonic for a set of character sets. It can be `iso-8859-n` where the suffix is an integer in the range 1 to 16, and the character sets used are equivalent to the given ISO 8859 character set. It can be `iso-2022-jp` which is equivalent to the character sets used in that encoding. If the value of the parameter is not one of these, then `iso-8859–1` is assumed.

ppmhs:ftbp

This generates a file-transfer-body-part. The data part is taken from the canonical data from the input component. The parameters use a variety of attributes from the source for items like the filename, as for `ppmixer:genericmime`. These value can be overridden by filter parameters.

application-reference
: sets the application reference object identifier. The value should be in numeric OID format.

single-asn1-type
: If present, then the external for the data uses the single-asn1-type CHOICE for the data external. Otherwise the octet-aligned choice is used.

no-message-reference
: If set, no message reference is set in the parameter, even if the source makes a value available.

ppmhs:ia5-text

This generates an ia5-text body part from the input canonical text data (in UTF-8). If has the parameter `AllowInvalidCharacters`, which if it has the value true allows non-ASCII characters in the body. If this is not set, then non-ASCII characters are converted to '?'.

ppmhs:teletex

This generates a teletex body part from the input canonical text data. Characters which cannot be represented in one of the character sets allowed in TeletexStrings are converted to '?'. There is one parameter, `PageBreakAtStart`, which if set to `true`, will cause a page break (CR FF) to be inserted at the start of the body.

ppmhs:bilateral

Takes the canonical data, and from it generates a bilaterally-defined body part. This filter has no parameters.

ppmhs:heading-modify

R15.0 and later] Modifies an IPM heading. The parameter notifications alters the notification requests in all recipient specifiers. takes a value which is a comma separated list from:

none
: remove all notification requests.

nrn
: add Non-Receipt Notification requests.

rn
: add Receipt Notification requests (implies nrn)

ipm-return
: add IPM Return request

ppmhs:p772-heading-upgrade

[R15.0 and later] Upgrades an IPM heading for use in a P772 (Military) message.

IPM identifiers are modified to make them conform to the requirements of STANAG 4406.

The primary-precedence heading extension is added to the heading. The value of the precedence is derived from the envelope priority of the message and any military-messaging priority qualifier present.

If the heading has copy recipients and/or blind-copy recipients, then the copy-precedence extension is added to the heading, using the same value as the primary-precedence.

The `extended-authorization-info` extension is added. The date/time used for it is message's submission time taken from the trace information in the envelope.

6.2.3.3 MIXER filters

MIXER filter are used when converting between Internet and X.400 content types.

ppmixer:envelope

Used to put heading fields derived from the X.400 message envelope into an Internet message content. It takes the parameter make822. If this is present, then the standard, required, RFC fields are generated from the message envelope. This option is used when the X.400 content is being wrapped in a single MIME body, and so there is no X.400 heading to be used for these fields.

ppmixer:ipms2rfc

Converts an IPM heading to an Internet message heading. It has parameters:

external
: Sets the domain for address normalization

internal
: Sets the domain for address normalization

show-notif-req
: Add comments to recipient addresses to display the notification requests in the IPM heading

charsets
: Specify the character sets which can be used with MIME encoded words (RFC 2047) when converting teletex strings in the subject or free form names.

fold
: Specify the line length at which folding will be performed.

convert-notif-req
: [R15.0 and later] Generate a Disposition-Notification-To field in the output. If the value is `@force@` , then the field is always generated; if the value is `@upgrade-nrn@`, then if any recipient specifier contains at least an NRN request, then it is generated. Otherwise it is generated if any recipient specifier contains an RN request.

ppmixer:rfc2ipms

Converts an Internet message heading into a IPM heading. It has parameters:

external
: Sets the domain for address normalization

internal
: Sets the domain for address normalization

use-mixer-ddas
: If set, put into the user element of the this-IPM IPM Identifier RFC heading fields defined by the PP variable `mixer_fields`.

make-p772
: Ensures that those P772 heading extensions which are required according to STANAG 4406 are present, if not present as a result of mapping MMHS heading fields. Values are taken from the message's envelope.

notifications
: [R15.0 and later] Control the notification requests generated if there is a suitable `Disposition-Notification-To` field in the source heading. The value is a comma separated list selected from `@nrn@`,`@rn@`,`@ipm-return@`. Note that `rn` implies `@@nrn@`. The default is all requests. If the values include `@force@`, then the notification requests are added even if there is no `Disposition-Notification-To` field in the source heading.

ppmixer:multipart2ipms

Converts a MIME multipart into a forwarded message body. If the parameter p2only is present, then only elements appropriate for IPMS 1984 (P2) are generated.

ppmixer:genericmime

Generates a standard MIME body from the source canonical data. It can take a number of parameters to define the attributes of the MIME body. Note that as the data can be arbitrary, the `content-transfer-encoding` is set to binary. Unless binary MIME content is being output, this filter should be followed by `ppmime:encode`.

Parameters:

Type
: Sets the MIME type

Subtype
: Sets the MIME subtype

parameters
: Has a value which is a space separated list of parameter names, which are added to the Content-type field of the MIME header.

paramname
: Gives the value for one of the parameters whose names are listed in `parameters`.

Disposition
: Gives the disposition value for the Content-disposition field. If omitted, and other such fields are included, then defaults to `attachment`.

filename,size,creation-date,modification-date,read-date
: Attachment file values for Content-disposition.

Description
: Value for Content-description field

Id
: Value for Content-ID field

norfc2231
: If set, then RFC 2231 encoding is not used for a filename parameter; MIME encoded-words are used for non-ASCII characters, and the value is not wrapped. If the value is an integer, then this value gives the maximum length of the parameter value. The default value is 600. This is to support interworking with Exchange 2003.

If these parameters are not set, the values can be obtained from the source component. For instance, Description and the file values can be available from a file-transfer-body-part.

ppmixer:tox400bp

Generates the application/x400-bp MIME body which is used to encapsulate an arbitrary X.400 body part

ppmixer:fromx400bp

Generates the X.400 body which is found in an application/x400-bp MIME body.

ppmixer:ftbp2mime

Extracts data and parameters from a file-transfer-body-part. These are presented in a form suitable for ppmixer:genericmime. It takes no parameters.

ppmixer:mime2ftbp

Generates a file-transfer-body-part from MIME data, and generates the body part parameters from the MIME body header fields.

ppmixer:fromharpoon

Generates a MIME body from an X.400 body which is a ‘HARPOON’ encoding of a MIME body (the encoded MIME body within an ia5-text X.400 body-part).

ppmixer:toharpoon

Encapsulates an arbitrary MIME body within an X.400 ia5-text body (‘HARPOON’ encoding).

6.2.3.4 Textual Security Label Filters

These filters work in conjunction with the security policies to extract and insert filters in components of the message. Note that the filters are invoked when the corresponding component is processed. So, you cannot insert a label in a component which is extracted from another component processed after this component.

Common insertion filter parameters

The insertion filters have a number of common parameters:

security-policy
: The name of the security policy to be used. If not specified, then `default` is used.

verify-label
: Verify the label using the policy. If this fails, and the policy has a default label, then the default is substituted.

convert-label
: Convert a label in a different policy to a label in the policy. If this fails, and the policy has a default label, then the default is substituted.

variant
: Specify conversion of the label to the named variant within the given policy.

need-policy-id
: If set and the label has no policy ID, then the default policy ID from the security policy is added to the label.

pplablib:labelinsert

Insert a text label into an Internet message heading. There are these parameters:

xheader-format
: Inserts a heading field containing security label information. The value of the parameter is a format for the whole field, including the field name.

subject-format
: Inserts a text label into the existing subject of the message. The value of the format is the whole of the field, including the field name. The data from the original subject can be included within the formatted field.

history-fieldname
: Insert heading using the given fieldname with text for the original governing label for the message. This is only inserted if the label has changed. It is only used in conjunction with `xheader-format`.

In addition, the filter takes standard parameters for label insertion.

The format can contain format specifiers which are used to include text values within the output string. The label values are normally from the markup information from the security policy. The specifiers are:

%b
: Generate the whole label as the base64 encoding of the BER encoding of the label

%d
: The data from the original subject (for the subject-format only)

%e
: Generate the value of an SIO-Label field using ESS label format for the binary part %c

%x
: Generate the value of an SIO-Label field using X.411 label format for the binary part

%l
: A text representation of the whole label

%c
: A text representation of the label's classification

%p
: The privacy mark from the label

%i
: The policy ID from the label

%g
: The security categories from the label

%%
: A percent sign

The text is generated from the security policy configured for the filter, if any is available. If there is no policy, some best effort is made to generate text.

The label field specifiers can be modified with characters between the '%' and the final character. The modifiers control where the text comes from, and for the case in getting the text from the security policy in force, which markup location is to be used.

#
: Use raw text rather than policy information

$
: Use the API rather than policy information (Not Yet Implemented)

<
: Use the document start location

>
: Use the document end location

-
: Use the page top location

+
: Use the page bottom location (the default)

Note: < and > must be escaped when placed in XML as they are 'special'. Use the entities `<` and `>`

There are '\' escapes which can be used:

\n
: adds a newline

\t
: adds a tab

\\
: adds a backlash

newlines and tabs in the format value itself are ignored.

pplablib:labelextract

Extracts a label from an Internet message heading, and makes it available, for example, for the X.400 envelope or inclusion as an ESS label in an S/MIME signed message. Parameters:

regexp
: A regular expression. Most Perl regular expression constructs are supported. Named fields are not supported.

replace
: The replacement text. This can use $<n> items to refer to the matched string or substrings of the input.

field
: The name of a heading field. The whole of the field value is used to determine the label.

base64
: Read the `base64` encoded BER encoded X.411 label.

attempt-base64
: First attempt to read the field as for base64, if that fails the string is checked as a normal text label.

catalog-lookup
: Use the value of the field to find a label in a label catalog. The associated `security-policy` must have a label-catalog attribute.

security-policy
: The name of the security policy to be used. If not specified, then `default` is used.

priority
: Specifies the priority to be used for the label found. This can be used to override the "first label found applies" rule which applies by default.

One of `base64`, `field` or both of `regexp` and `replace` should be specified. `base64` does not need a security policy, but the others do.

The combination of the regular expression and the replacement text functions as the `s/<re>/<replace>/` operator in Perl, or the `regsub` operator in Tcl.

The conversion of text to a security label is a two-stage process. If the field as a whole matches the regexp, or the field name matches the value of `field`, the either the replacement text or the field value, respectively, is passed to the security policy for conversion to a label. If this fails, but a policy ID and classification can be determined, and the security policy has an associated label catalogue, then a fallback label can be found in this.

Example of label extraction

[Note: this describes R15.2v1 and later]

This example is in the context of MIXER conversion of a message. It looks for suitable labels in:

- within the subject field
- a base64 encoded label in a X-X411 field
- within a textual X-header field

Different priorities are applied to these, so the order in the heading is not important.

The converter for the Internet message heading becomes

```
<convert type="header" action="convert" cost="2">
  <filter command="pplablib:labelextract">
    <param name="regexp">
 (?i)(^subject:.*\[Classification:([^]]*)\])
    </param>
    <param name="replace">$2</param>
    <param name="security-policy">uk-only</param>
    <param name="priority">3</param>
  </filter>
  <filter command="pplablib:labelextract">
    <param name="base64">X-X411</param>
    <param name="priority">5</param>
  </filter>
  <filter command="pplablib:labelextract">
    <param name="field">X-classification</param>
```

```
        <param name="security-policy">uk-only</param>
        <param name="priority">4</param>
      </filter>
      <filter command="ppmixer:rfc2ipms">
        <param name="notifications">rn</param>
        <param name="use-mixer-ddas"/>
      </filter>
    </convert>
```

The first filter extracts textual labels from the subject, from an string like

```
Subject: This is an important message [Classification: TOP SECRET]
```

The second filter extracts the base64 label. The third extracts a text label from a field like:

```
X-Classification: Restricted
```

These have to process the heading before it is converted to an IPMS heading by the MIXER filter.

pplablib:flotinsert

Insert a "first line of text" (FLOT) label into the data of a text body. This filter will only do this the first time it is invoked for a message. Parameters:

format
: Expanded the first line of text. The format uses the same mechanism as `pplablib:labelinsert`.

dont-duplicate
: If the line generated from the format matches the existing first line, then the extra line is not added. The comparison ignores case, initial and final white space, and internal white space is treated as a single space.

history-prefix
: Insert a second line of text comprised of the prefix and text for the original governing label. This is only generated if the label has changed.In addition, the filter takes standard parameters for label insertion.

For a message, only the first use of this filter will attempt to add a FLOT.

Example of FLOT insertion

The comparison for the first line works with UTF-8, so filters are used to convert character data to UTF-8, and then the data is re-encoded.

```
    <convert type="body" action="convert" cost="6">
      <match name="Type">text</match>
      <match name="Subtype">plain</match>
      <filter command="pplablib:flotinsert">
        <param name="format">Classification: %l</param>
        <param name="security-policy">uk-only</param>
        <param name="convert-label"/>
        <param name="dont-duplicate"/>
      </filter>
      <filter command="ppmime:encode">
        <param name="Encoding">quoted-printable</param>
      </filter>
    </convert>
```

pplablib:flotextract

Extract a FLOT label from the data of a text body. Parameters:

regexp
: A regular expression. Most Perl regular expression constructs are supported. Named fields are not supported.

replace
: The replacement text. This can use `$<n>` items to refer to the matched string or substrings of the input.

prefix
: Prefix to the text to be used for recognition.

suffix
: Suffix to the text to be used for recognition.

lines
: The number of lines at the start of the body which are checked for a label (i.e. matching the regexp or the prefix and suffix). The default value is one, i.e. only the first line is checked.

security-policy
: The name of the security policy to be used. If not specified, then default is used.

priority
: Specifies the priority to be used for the label found. This can be used to override the "first label found applies" rule which applies by default.

Either `regexp` and replace should be specified, or one or both of `prefix` and `suffix`. The regular expression mechanism works in the same way as `labelextract`. Using `prefix` and `suffix`, any white space at the start and end of the line is ignored. Then, if the line starts with the prefix and ends with the suffix (treated as zero-length strings if not configured), then the text between them is used for finding the label. The prefix and suffix are matched without regard to case.

The conversion of the text to the label functions in the same way as `labelextract`.

Examples of FLOT extraction

The context here is for SMTP to SMTP messages. text/plain body parts are inspected. The filter passes the canonical data, with the transfer encoding removed, so it needs re-encoding. The converter using for them is as follows:

```
<convert type="body" action="convert" cost="6">
  <match name="Type">text</match>
  <match name="Subtype">plain</match>
  <filter command="pplablib:flotextract">
    <param name="regexp">
       Classification: (.*)
    </param>
    <param name="replace">$1</param>
  </filter>
  <filter command="ppmime:encode">
    <param name="Encoding">quoted-printable</param>
  </filter>
</convert>
```

This converter does the same, using a prefix:

```
<convert type="body" action="convert" cost="6">
  <match name="Type">text</match>
  <match name="Subtype">plain</match>
  <filter command="pplablib:flotextract">
    <param name="prefix">classification:</param>
  </filter>
```

```
      <filter command="ppmime:encode">
        <param name="Encoding">quoted-printable</param>
      </filter>
    </convert>
```

pplablib:envinsert

This filter inserts the current message label in the message envelope, for the recipients of the message which is being converted. This is used when some check or change to the label is required. For instance, if label conversion is to be performed.

It takes just the common label insertion filter parameters.

pplablib:conflictfield

This filter adds a heading field to an SMTP heading to record information on label conflicts found in the message. It has one mandatory parameter, `fieldname`, which sets the name of the field to insert.

6.3 MIXER Content Conversion

6.3.1 Internet to X.400

This conversion basically follows RFC 2156 and RFC 2157, with some additional features:

- Standard messages are converted to X.420 IPMs (Inter-Personal Message).
- DSNs (Delivery Service Notifications) are converted to X.400 Reports if possible [from R15.0].
- MDNs (Message Disposition Notifications) are converted to X.420 IPNs (Inter-Personal Notification) [from R15.0].

6.3.2 X.400 to Internet

- IPMs are converted to messages
- IPNs are converted to MDNs [from R15.0]
- Reports are converted to DSNs.

MDNs and DSNs carry a text body-part intended to be read by the message originator. The text body which is generated by M-Switch can be configured.

6.3.3 Correlation

A sender of an X.400 message may wish to correlate received Reports and IPNs with their sent messages. A sender of an Internet message may wish to correlate received DSNs and MDNs with their sent messages.

M-Switch supports the standard means for such message correlation across the MIXER gateway:

- The MTS Identifier for correlating X.400 Reports with messages.
- The IPM Identifier for correlating X.400 IPNs with messages.
- The SMTP ENVID extension, used to correlate DSNs with messages.

- The Internet Message-ID field, used to correlate MDNs with messages.

The MIXER specification converts these to fields in the other type of message such that the item (Report, IPN, DSN, MDN) which is returned, when converted will carry the correlating value in the correct place for the item.

However, there are some issues with this.

- Internet Message-IDs can be too long to be carried in the user-relative-identifier of an IPM Identifier.
- Some Internet message senders do not use ENVID.
- Some message correlation software does not use the standard fields in DSNs or MDNs for the correlation.
- To allow for this, M-Switch has some additional features to aid such correlation.

6.3.4 Specifying Internet Heading Fields to be transmitted

If an Internet message has an ENVID, then it is assumed that the sender uses this for correlation. This is encoded in the X.400 message's content-correlator.

If there is no ENVID, then some of the Internet message's heading fields can be included in the content-correlator. This should be returned in the report, and the Report to DSN conversion can use the enclosed information.

It is possible to specify the fields to be copied. There is a PP variable (configured via the Advanced tab) called `mixer_fields`. If set, its value is a spaced-separated list of field names to be used. If it is wanted to truncate the field value, then follow the field name with the maximum length in parentheses. This is an example as it will appear in the `mtatailor.tai` file:

```
set mixer_fields="Thread-index Message-ID Subject(20)"
```

If this is not specified, a default list is used, corresponding to:

```
set mixer_fields="Subject Message-ID Date"
```

Note that the content-correlator has a maximum length of 512 characters, and the encoding includes the field names. The field names match ignoring the case of the letters. The fields are copied in order until the content-correlator is filled.

On receipt of a Report containing a returned content-correlator which contains such encoded fields, M-Switch will use this information. It will make it available to the text body generation. If the information contains a `Message-ID` field, then the value of this is used to add an `In-Reply-To` field to the outer DSN heading. If the information contains a `Thread-Index` field, then from its value a child thread-index value is generated and used as in a `Thread-Index` field in the outer DSN heading.

6.3.5 Specifying fields for MDN correlation

When an Internet message is converted to a IPM, the `Message-ID` is used to generate the IPM Identifier. This is returned as the subject-ipm field in an IPN, and converted back to Internet form, used for the `Original-Message-ID` field in the MDN generated from the IPN. To work round the problem of long message-ids, and also to provide other information used in MDN generation, it is possible to store Internet message heading fields in the IPM Identifier. This storage uses DDAs within the user address part of the IPM Identifier.

The same PP variable is used to configure which fields are included. Note that as the information uses a restricted character set, the total amount of information that can be conveyed is somewhat less than in the content-correlator.

When an IPN is converted into an MDN, the subject-ipm IPM identifier is checked for this information. It is used in the same way as for DSN generation for the text body and outer MDN heading, with the additional use of a message-id from this information overriding any original-message-id value derived from the user-relative-identifier of the subject-ipm.

This use of DDAs is not automatic, it should be configured in the converter which converts the outer Internet heading into the IPM heading. This is done using the parameter `use-mixer-ddas` to the `ppmixer:rfc2ipms` filter, e.g.

```
<filter command="ppmixer:rfc2ipms">
  <param name="use-mixer-ddas"/>
</filter>
```

6.4 MIXER Address Conversion

The conversion between X.400 OR-addresses and Internet e-mail addresses is normally performed in accordance with the procedures defined in RFC 2156. These are based on finding the longest match in the mapping information, and then applying an algorithm to the unmatched part of the address. The mapping information can be held in the Directory, in address mapping trees, or in tables (the `or2rfc` and `rfc2or` tables).

There are some enhancements to the standard procedures:

- The string format used supports the universal string addressing attributes.
- The character used as the separator for the mapped personal name can be changed.
- Conversion to and from the mapped personal name form can be suppressed.
- The string representation of the OR-address in the internet address can have spaces replaced by a non-space character.
- The mapping can be configured to search for individual address conversions.

6.4.1 Individual Address Mapping

This uses LASER-like searching in order to find an entry which contains the address which needs to be converted, and an attribute within that entry gives the address resulting from the conversion.

6.4.1.1 Configuring per-user mapping

The configuration of which addresses are configured for the per-user conversion uses the same information that is used for the standard address mappings.

For Directory-based information, for a node in an address conversion tree, there is a choice of setting the Directory profile to be used. This choice actually sets the name of the LASER table to be used, although the term “Directory profile” is used throughout MConsole. A default Directory profile is always set up. Any other name is used as the table name prefix, and the table called `prefix-laser`. This can be applied to the tree at the root for an address type, which means that the lookup will apply to all addresses of that type.

For table-based information, the table entry has a value which the string for the LASER table, as in the directory based information.

6.4.1.2 Configuring LASER tables for per-user address mapping

The table configures LASER lookup in the same way that is used for LASER routing M-Switch Administration Guide). If there is a conflict between the use for routing and address mapping, then use two LASER tables, distinguished by using a different prefix. The lookup uses the same basic LDAP configuration values as routing.

There are additional table entries to control the address mapping lookup:

mixer-internet-atts
: Defines the attributes used to construct the search filter when looking up an Internet address.

mixer-x400-atts
: Defines the atttributes used to construct the search filter when looking up an X.400 address.

mixer-filter-internet
: Defines an LDAP search filter which is combined with the constructed filter using an ‘AND’ when looking up an Internet address.

mixer-filter-x400
: Defines an LDAP search filter which is combined with the constructed filter using an ‘AND’ when looking up an X.400 address.

The filters are used to qualify the filter constructed from the attributes and the address being looked up.

The table entry `laser-atts` is used to specify the attributes to be read from the entry. Note that this entry is used for all lookup types using this LASER table. The use of each attribute specified in `laser-atts` needs to be configured using a table entry with a key which is the attribute type. The value of the entry specifies the use of the attribute type. This can be one word or two words separated by a space. Using two words enables the same attribute type to be used for routing and for address mapping.

There are two attribute meanings for the per-user address mapping:

mixer-internet
: The attribute holds the Internet version of the user’s address.

mixer-x400
: The attribute holds the X.400 version of the user’s address.

If the table entries are not present, then there are some defaults, equivalent to:

```
mixer-internet-atts: mail
mixer-x400-atts: mhsORAddresses
mail: mixer-internet
mhsORAddresses: mixer-x400
```

Note that if you specify `laser-atts`, you must explicitly list all attributes which are to be looked up for routing and address mapping. And you must have explicit table entries for each attribute specified in `laser-atts`.

Chapter 7 M-Switch ACP 127 Operating Signals

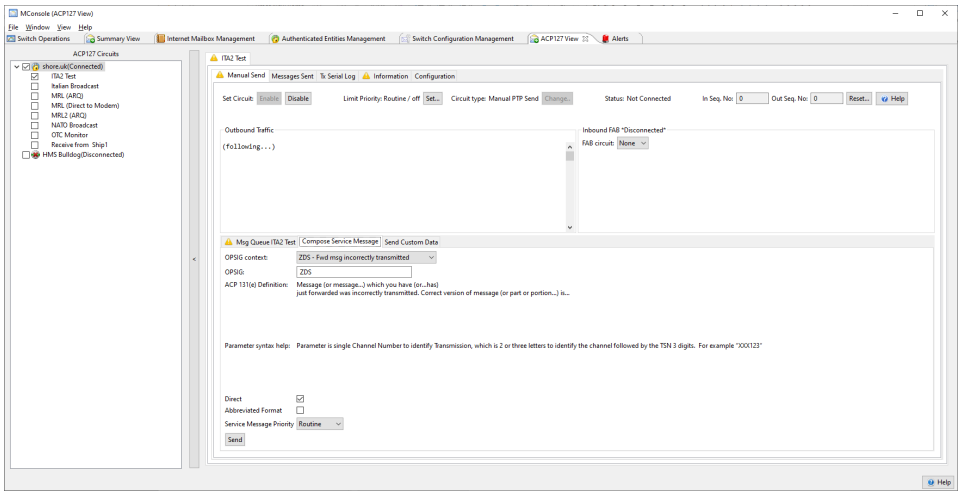
This Chapter describes M-Switch handling of ACP 127 Operating Signals (OPSIGs).

7.1 M-Switch Architecture and Service Message Support

M-Switch provides ACP 127 support as an ACP 127 gateway, converting protocol to STANAG 4406 or SMTP. It is intended to work as a fully automatic system wherever possible. Signals are often included in ACP 127 messages for automatic control purposes, including automatically generated service messages. Messages will be generated by protocol conversion or ACP 127 relay.

MConsole, the management UI for M-Switch provides the circuit operator with a UI for generating a service message and sending it on the selected circuit. This enables an operator to generate any service message. A drop down list of commonly used operating signals is provided, but any signal may be entered. This allows any operating signal to be sent, and so any signal can be sent manually.

Figure 7.1. MConsole ACP127 View Manual Send.



On an actively monitored ACP 127 circuit, the operator may well observe incoming service messages on the activity trace UI. Service messages directed to the local RI, apart from those handled automatically, are converted to SMTP or STANAG 4406, in a manner that does not lose information. This enables an operator to view service messages in a standard mailbox and to handle any signal manually.

7.2 Operating Signals

The following table considers each OPSIG handled by M-Switch. Note that OPSIGs are handled in three ways.

1. Using the OPSIG directly in a service message. This happens where the OPSIG conveys information or provides an answer.
2. Using INT OPSIG in a service message. This is a question, to which an answer is expected.
3. Use of the OPSIG in a non-service message. The descriptions here refer to service messages, except were explicitly noted otherwise. Where the INT variant is used, this is explicitly noted

The descriptions here refer to service messages, except where explicitly noted otherwise. Where the INT variant is used, this is explicitly noted.

Table 7.1. OPSIG: QRT

Signal	Description	Generation	Reception	Notes
QRT	Shall I stop sending? / Stop Sending	For a receiving system, the OPSIG sending UI the QRT sending drop-down notes that this will request peer disable. There is also UI to disable the peer from sending which will generate a QRT.	On receiving QRT, a sending system will automatically disable outbound transmission.	The message has a three letter channel designator to identify that circuit that should be closed or open.

Table 7.2. OPSIG: QRV

Signal	Description	Generation	Reception	Notes
QRT	Are you ready/ I am ready	For a receiving system, the OPSIG sending UI the QRV sending drop-down notes that this will request peer enable. There is also UI to enable the peer from sending which will generate a QRV.	Manual	

Table 7.3. OPSIG: ZFX

Signal	Description	Generation	Reception	Notes
ZFX	Channel Number is Open	Report that channel number is open. Receiver of ZFX may retransmit indicated message This is a report that a number in the channel number sequence has not been seen. (ACP127 para 433). It is appropriate on a sequenced channel between two stations. If a broadcast receiver is missing a channel number, then it does not know if the message is for it or for other stations. M-Switch can configure use of ZFX in broadcast receiver as alternative to ZDK (ZDK recommended)	Node receiving messages will identify missing message and send service message with ZFX.. Node receiving ZFX will retransmits requested message if configured.	Used for broadcast

Table 7.4. OPSIG: ZDK

Signal	Description	Generation	Reception	Notes
ZDK	Will you repeat message	Requests retransmission Message receiver will use INT ZDK as a request for sending again message identified by channel number.	Retransmits requested message Then the message which is resent has ZDK <channel number> in FL4. This is the default for broadcast in M-Switch.	Recommended for use in broadcast

Signal	Description	Generation	Reception	Notes
			The broadcast message listings also include it in the details of the message, so that a station to which the message is addressed which has seen the original but not the repeat need not request the resending of the repeat.	

Table 7.5. OPSIG: ZFQ

Signal	Description	Generation	Reception	Notes
ZFQ	Duplicate Detected	Optionally generated (configuration choice)	Manual	No sensible automatic action on reception

Table 7.6. OPSIG: ZFG

Signal	Description	Generation	Reception	Notes
ZFG	This message is an exact duplicate of a message previously transmitted and is to be delivered to all appropriate addressees served by the receiving communications facility.	Optionally used on broadcast to warn of duplicates (configuration choice – default is to discard duplicates)	Service message sent to operator	Anticipated use is specialized Italian situation (OTC). It appears in FL1 and is used rather than using ZDK to indicate the message of which this is an exact repeat.

Table 7.7. OPSIG: ZFT

Signal	Description	Generation	Reception	Notes
ZFT	Message...received without channel number(s) (or station serial number) following message bearing channel number (or station serial number) ...Message released.	Optionally generated when message has no sequence number (configuration choice)	Service message sent to operator	No sensible automatic action on reception

Table 7.8. OPSIG: ZES1/ZES2

Signal	Description	Generation	Reception	Notes
ZES1/ZES2	Your message...has been received... 1. Incomplete; 2. Garbled. Request retransmission.	Manual	When receiving a ZES2 with a channel number, M-Switch will be extended to automatically retransmit the requested message. The retransmission will have ZFG + channel number in the message instructions	ZES signal will always be initiated manually in response to operator detected partial failure (complete failures will not be detectable). This is supported in MConsole with drop down. On many circuits where garble is possible, channel

Signal	Description	Generation	Reception	Notes
			and be transmitted with a new channel number Other situations handled manually	numbering is not used, so automatic handing on reception is not possible.

Table 7.9. OPSIG: ZNO

Signal	Description	Generation	Reception	Notes
ZNO	Unable to decrypt message...	Manual	Manual	No sensible automatic handling

Table 7.10. OPSIG: ZUI

Signal	Description	Generation	Reception	Notes
ZUI	Your attention is invited to...	Manual	Manual	No sensible automatic handling

Table 7.11. OPSIG: ZBZ

Signal	Description	Generation	Reception	Notes
ZBZ	What is the printing acceptability of my signals (or those of...)? The printing acceptability of your signals (or those of...) is... 1. Unacceptable - totally corrupt; 2. Unacceptable - very corrupt; 3. Unacceptable - partly corrupt; 4. Acceptable - occasionally corrupt; 5. Acceptable - no corruption.	Manual MConsole OPSIG UI includes INT ZBZ (request message printing quality) to make this easy to select.	Manual by default Provides an option on receive circuit (for use only on reliable circuits) to automatically send ZBZ5 when the message is identified (i.e., on a circuit with channel numbers).	ZBZ is generally used on circuits where message garble is possible. Generation of ZBZ needs an operator to assess which value is needed. ZBZ is also used on reliable channels (TCP, land line, COSS) to check message receipt. Initiation (INT ZBZ) will always be operator driven. For a reliable link, ZBZ5 can be automatically generated.

Table 7.12. OPSIG: ZAN

Signal	Description	Generation	Reception	Notes
ZAN	Transmit only messages of and above precedence...	Generated at operator request in MConsole precedence management UI	Automatic Precedence Control following signal Local operator alerted. Local operator requests take priority over ZAN	

Table 7.13. OPSIG: ZIC

Signal	Description	Generation	Reception	Notes
ZIC	What is (are) station serial number(s) or channel number(s) of last message(s) you transmitted to me (or to...)?	Sent in response to a ZID See ACP 127(G) para 412 for ZIC/ZID	Discard	

Table 7.14. OPSIG: ZID

Signal	Description	Generation	Reception	Notes
ZID	What is (are) station serial number(s) or channel number(s) of last message(s) received from me (or from...)?	Automatic generation (optional) at configurable time after no traffic has been received	Generate a ZIC	

Table 7.15. OPSIG: CHANNEL CHECK

Signal	Description	Generation	Reception	Notes
CHANNEL CHECK	Manual channel Check See ACP127G para 412e/f	MConsole provides a 412e compliant test message (this is not a OPSIG) to make it straightforward for an MConsole operator to send a correct CHANNEL CHECK message. On receipt of the returned message, it will be delivered to the address associated with the local operator.	On peer receipt of CHANNEL CHECK message, this will be routed back to the sender.	412e defines a special CHANNEL CHECK message. Note that CHANNEL check is a special message and is not a service message. It is addressed to the local address and sent over the link so that it is echoed back.

Table 7.16. OPSIG: CHANNEL CONTINUITY

Signal	Description	Generation	Reception	Notes
CHANNEL CONTINUITY	Process described in ACP127G para 412 This uses ZIC/ZID	Automatic or Manual	Automatic or Manual	ZIC/ZID will usually be configured to be automatic. ZIC/ZID can also be used manually with MConsole

Table 7.17. OPSIG: R Z

Signal	Description	Generation	Reception	Notes
R Z	Flash Ack	Automatically generated on FLASH message (configuration choice to do this). Follows ACP127G point 151 b	Cease sending repeats of messages	This is a prosign, not an operating signal

Table 7.18. OPSIG: INT R Z

Signal	Description	Generation	Reception	Notes
INT R Z	Cannot find any information on this	Manual	Manual	

Table 7.19. OPSIG: OSL

Signal	Description	Generation	Reception	Notes
OSL	Can you acknowledge receipt?	Manual	Manual	Automatic handling does not seem appropriate

Table 7.20. OPSIG: ZAH

Signal	Description	Generation	Reception	Notes
ZAH	Unable to relay message...in present form 1. Not in prescribed format; 2. Format lines...incorrect; 3. No on-line facility available; 4. Call signs not encrypted; 5. Text not encrypted). We file. Transmit correctly prepared message to all addressees (or to...).	Manual	Manual	Automatic handling does not seem appropriate

Chapter 8 Security

M-Switch has a rich set of Security features. This chapter describes the different features, what they do and how they are configured.

8.1 Overview

M-Switch has a number of features relating to security:

- Authorization can control the delivery of messages by blocking messages. This is done through Authorization Rules. The rules can include the results of checking the S/MIME signatures on messages, and for labelled messages, a security policy can be applied with the clearance of the recipient, peer MTA or channel.
- Internet messages can be signed by the MTA using S/MIME signing.
- X.400 messages can be signed by the MTA in accordance with STANAG 4406 ed 2.
- Internet messages can be signed using DKIM (DomainKeys Identified Mail, RFC 4871).
- Internet and X.400 messages can be encrypted and decrypted using S/MIME encapsulation.
- There are various ways in which security labels are processed by M-Switch.

Signing and encryption can include messages being converted in a MIXER gateway. These are performed as a part of content conversion.

A useful set of examples of signing and encryption configuration are provided in the signing/encryption examples.

A useful set of examples of label extraction, verification and insertion configuration are provided here: label extraction, verification and insertion examples.

8.2 Security Labels

8.2.1 Background

Security labels are generically a means for identifying the sensitivity of information and therefore restricting its distribution. Labels used for computer based information are often based on those defined in X.411 and ESS labels defined in RFC 2634. These have the following components (and so are often described as “structured labels”):

- *Policy ID*. This is an object identifier, which identifies the authority which defines labels with this ID. In some cases, the ID represents the ownership of the information.
- *Classification*. This is an integer value, which gives the basic secrecy level. However, higher numerical values are not necessarily more secret.
- *Categories*. These are optional, and if included qualify the classification. Each value is a pair of an object identifier and a value whose type is associated with object identifier. The valid IDs and values are specified by the authority.
- *Privacy Mark*. This is a text string. It is not often used.

Associated with security labels are other data items. A *Security Policy Information File* (SPIF) is structured data which defines for a given policy ID the valid classifications and security categories. It also can define strings to be associated with labels, which are used for mark-up of data for human reading. It can define equivalent policies, which enables labels defined by a different authority to be associated with labels defined in this SPIF. It also defines how the ‘Access Control Decision Function′ (ACDF) is to be applied.

A *Clearance* is a data object very similar to a label, except that the classification is replaced with a set of classifications. Clearance values are assigned to people and other entities which may access labelled information. The ACDF takes a label and a clearance and establishes whether access should be granted.

Sometimes information is labelled using a text string. This string will often be derived from the mark-up strings for the label and its components. When the string is at the start of a body of text, it can be known as a "First Line of Text" (FLOT).

For messages, labels may be found:

- In the envelope of X.400 messages
- An ESS label within signature information for an S/MIME (Cryptographic Message Syntax) message.
- A structured message format within an Internet message heading field
- Text in an Internet message heading field, sometimes known as an X-header.
- A FLOT

8.2.2 M-Switch Label Handling

M-Switch can:

- Extract labels from the message envelope and content
- Insert labels into the message envelope and content in various forms
- Use labels in authorization in conjunction with a SPIF and clearance associated with a channel, peer MTA or user.

8.2.2.1 Label extraction

Label extraction can be done when the message is scanned on arrival and by the shaper channel when performing content conversion.

X.411 envelope labels are always found, no configuration is required for this. For message content scanning on arrival and in the shaper channel, ESS labels in S/MIME signed messages are always extracted, as are the labels in SIO-Label heading fields. To extract other kinds of label requires explicit configuration of specific shaper filters for this purpose. In the shaper channel, the latter mechanism will extract the labels as the content's components are processed for conversion. Note that this means some of the content's components may already have been processed when the label is found.

By default, “the” label for the message is taken to be the first label found. On message arrival, this is label is associated with the message envelope. This label is available for the shaper channel before any conversion process is performed. It is possible to associate a priority with a source of labels, which will override the “first used” principle.

The conversion of text labels to a proper structured label requires the use of security policy information.

8.2.2.2 Label Insertion

When inserting labels it is possible to:

- Use a default label if no label is available
- Verify the label against a security policy
- Convert the label using a security policy
- Add a policy ID to the label, if it is missing

If verification or conversion fail, then any default label will be used in place of the failed label.

When a label is found, it can automatically be used as an X.400 envelope label. In addition, there is a shaper channel filter which can be used to establish an envelope label (for the message being converted) when verification or conversion is required.

When a signed message is generated, any label for the message is automatically included in the signed attributes of the message.

Other label types are inserted through the use of shaper filters. Text labels are generated using the markup strings from SPIFs in the security policy.

8.2.2.3 Use in Authorization

The label found in the message on arrival can be used for authorization. The security policy used for this is called "acdf-policy". If this is not defined, then the policy called "default" is used. If neither is available, then the authorization fails.

8.3 Content Scanning

The content of a message may be scanned when the message arrives in the MTA. This can be:

- To extract information from an X.400 IPM or IPN for logging in the audit log
- To extract information from an Internet DSN or MDN for logging in the audit log
- To check the signature of a signed message
- To extract labels for use in authorization

If there is no need for any of these, then normally the content is not scanned. If it is desired to force the scanning, then set the PP variable `always-scan-content` to the value `true`.

Configuration

The actions taken on scanning are controlled by an XML configuration file called *submitscanconfig.xml*. A default copy is distributed, installed in *SHAREDIR/switch*. If modifications are to be made, then this file should be copied to *ETCDIR/switch*, and the copy edited. This will avoid changes being lost should the installed software be updated. However, when upgrading to a new release, check the new installed version for changes.

The configuration changes which you are likely to make relate to:

- Label extraction, involving the configuration of shaper filters
- The configuration of security policies for use in label label extraction and authorization checking using labels.

8.4 S/MIME Channel Configuration

The details are given in the relevant sections. Here is an outline.

8.4.1 Verification

- You should configure the channel for verification. A digital identity is not required, but trust anchors are required.
- The MIME exploder should be configured to ensure that the S/MIME bodies are recognized.
- You should decide if the S/MIME bodies should have the signed data extracted, through the use of suitable converter actions.
- Verification and extraction of X.400 signed messages requires that the content type `oid.1.2.840.113549.1.9.16.1.6` be '`exploded`'. To extract the content within the signed message, use the '`extract`' action for the converter for the content.

8.4.2 Signing

- You should configure the channel for signing. A digital identity is required, but other trust anchors are not required. This configuration is the same for signing S/MIME messages and for STANAG 4406 ed 2 signing.
- For S/MIME signing of Internet messages, you should decide how to sign, by setting the `sign` action on a converter for either a header or content. This converter may require additional configuration.
- For STANAG 4406 ed 2 signing of X.400 messages, the signing should be done for the output content type `oid.1.2.840.113549.1.9.16.1.6`, which identifies content which is CMS (Cryptographic Message Syntax). This output should have the same converters as for converting to P772 (`oid.1.3.26.0.4406.0.4.1`) with the exception that the converter for the content component should have the action '`sign`'. The converter takes a parameter `input-type` which has a value giving the content type for the content type to be signed. This can be the special value `p2orp22`.

8.4.3 Encryption/Decryption

- You should configure the channel for encryption and/or decryption. For decryption, a private key is required, corresponding to the public used by senders of messages to be decrypted. For encryption, a certificate containing a suitable public key is needed for each recipient (or peer MTA).
- When generating a triple-wrapped X.400 message, the output content type should be `oid.1.2.840.113549.1.9.16.1.6.` If the inner content type is to be P772, the converters should be suitable, and the '`input-type`' parameter on the converter should be `oid.1.3.26.0.4406.0.4.1`.

8.4.4 Label Extraction/Insertion

If an S/MIME message is verified, and it contains an ESS label, then it is automatically made available for the X.400 message envelope. If a message is signed, and a label is available either from the X.400 envelope or from label extraction, then it is included as an ESS label. If the source label is from a different security policy from the local policy, then the label is converted to the local policy, as long as the label's policy is an equivalent policy within the local security policy.

Insertion filters can be used to insert text labels into message headings and as first line of text. Extraction filters can be used to look for text labels and make the label available for the X.400 envelope or for an ESS label in a message being signed. If using extraction filters, you will need to configure the channel's `in-table`.

If a message is being signed and there is no label available from the message, then a default label can be specified for the shaper channel. This ensures that all signed messages are also labelled. If a security policy is specified for the channel, then the label must conform to that policy.

8.5 DKIM Configuration

DKIM signing is performed as part of content conversion. The supplied configuration already has a basic configuration set up for this. Normally, such signing is only applied to messages received via certain channels, such as a channel used for authenticated SMTP submission. The subtype-in for such channels is set to @dkim-sign@. This selects the converters required for the signing.

8.5.1 Shaper channel configuration

The signing is invoked by applying the ‘sign’ action to the ‘envelope’ component of the message. The converter used for this will normally have some match selectors to restrict the messages to which it applies.

8.5.2 DKIM configuration

There are a number of values which can be used to configure the signing. These are either configured as parameters to the convert within the shaper channel’s XML file, or can be configured as PP variables. The former take precedence.

dkim_private_key
: (Mandatory) The name of the file containing the private key to be used for the signing. This should be a PEM file containing the RSA key. If the filename is a relative pathname, then it is relative to *$(ETCDIR)*.

dkim_key_passwd
: The password for the private key, if required. This value can be encrypted using the server password mechanism. If in a PP variable, this happens automatically on creating the *mtatailor.tai* file from information in the directory.

dkim_selector
: (Mandatory) The selector for the key, which is the name used to construct the DNS name used to retrieve the key.

dkim_sign_alg
: (Mandatory) A string giving the signing algorithm and the hashing algorithm. Only two values are currently supported: `rsa-sha256` and `rsa-sha1.`

dkim_header_fields
: A colon-separated list of header fields to be used for the signing. If not included, a default set which includes the recommended fields is used.

dkim_signing_domain
: The domain deemed to be performing the signing. This is the most significant (right-hand) part of the DNS name containing the public key. If omitted, the local domain configured for the MTA (`loc_dom_site`) is used.

Here is an XML fragment for a converter for signing, where the mandatory values are parameters:

```
<convert type="envelope" action="sign">
  <param name="dkim_private_key">switch/dkim-private.pem</param>
  <param name="dkim_selector">selector</param>
  <param name="dkim_sign_alg">rsa-sha256</param>
  <match name="Content-subtype">dkim-sign</match>
</convert>
```

8.5.3 Table-based DKIM configuration

To allow support for multiple signing domains, each with their own DKIM settings, support for selection of parameters from a table, based on originator addresses, is also available. This is configured as follows:

A special converter parameter "dkim_signing_table" specifies the name of the table to be used. This can also be set as a PP variable.

An optional converter parameter (or PP variable) "dkim_address_fields" allows selection of the address field from the message being signed which is to be used as the key for table lookup. Any valid header address field can be selected (e.g. "from", "return-path"), plus the special value "rfc822-originator", which specifies the envelope originator address (i.e. the argument to the "MAIL FROM:" SMTP command. Multiple address specifications can be provided, as a colon separated list, with matches being tried in the order specified - e.g. "from:rfc822-originator" would first try a match on the From header field, and if that failed would try the envelope originator address. If no "dkim_address_fields" setting is present, the code falls back to use of rfc822-originator.

Support for exact address match, address domain and domain suffix matching is provided. This is performed automatically during table lookup: for example, a lookup of "tc@isode.com" would first attempt an exact match for "tc@isode.com", then "isode.com" and finally "com". If any of these lookups succeeds, then any absent parameters from the matched table entry will be set from the "default" entry (if present).

The table value field allows configuration of all of the existing DKIM parameters, as detailed below.

A "default" entry which can specify fallback values is supported.

The DKIM table is set up in the same way as existing tables and is handled with existing infrastructure, so it can be configured as a DIR, LINEAR, DBM or EMPTY.

The value field of a table entry consists of a set of comma-separated key=value pairs, with keys: K=<signing key file> P=<passphrase for key file if required> S=<DKIM selector> D=<signing domain> F=<header field list for DKIM algorithm> A=<signing algorithm>

If a given DKIM parameter is not configured via the table, the code will fall back to use of values from the Shaper configuration file, and if not set there, PP variables.

The default header field list is hard-coded as a colon-seperated list of the following fields: From, Sender Reply-To, Subject, Date, Message-ID, To, Cc, MIME-Version, In-Reply-To, References, Content-Type, Content-Transfer-Encoding, Content-ID, Content-Description

If the Signing Domain is not specified, it defaults to the value of the PP variable "loc_dom_mta".

The only supported Signing Algorithms are "rsa-sha1" and "rsa-sha256".

8.5.4 DNS configuration

For systems to verify the signatures, you will need to add a suitable TXT record to your DNS. The name used is <*selector*>`._domainkey.`<*signing-domain*>. The contents of

the record are described in RFC 4871. How you do this is out of the scope of this documentation.

8.6 Configuration of Cryptographic Services

M-Switch uses cryptographic services in various contexts:

- Signing messages
- Verification of signed messages
- Encryption and decryption of messages
- TLS in various contexts both as server and client
- Use of Strong Authentication

This section describes the use and configuration of the underlying cryptographic services. These are used in following:

- the use of mechanisms for the Cryptographic Message Syntax (CMS) in signing/verification and encryption/decryption
- the use of TLS by the SMTP server and Channel
- the use of TLS by the Corrector Channel

Other areas will move to use this in future releases. Even for CMS signing and verification, the older configuration data can be used, which is used to configure the new interface in a limited way.

Note: encryption/decryption is a separately activated feature.

8.6.1 Overview

The underlying library has two main aims. The first is to enable the use of different underlying cryptographic service providers, in particular:

- OpenSSL
- To hardware which uses the PKCS#11 API
- Microsoft CryptoAPI (CAPI), including access to hardware such as smart cards through this API

Currently only the OpenSSL backend is available.

The second aim is to have a Security Database which holds configuration items and other data for use by the cryptographic services. The crypto API can also access data in the Security database held by underlying cryptographic services. These are:

- Specification of trust anchors (TAs)
- Certificates
- Private keys
- Secret keys
- Configuration for certificate verification

There are two kinds of Security Database: ephemeral and permanent. The ephemeral database is used when the data and configuration information is held in different forms, and so the database, which is entirely within memory is loaded with the data when the

program starts. The permanent database is a simple, local, SQL database. It can be password protected, in order to keep sensitive information, such as private keys, encrypted. .

The Security Database can be accessed in one of two ways.

- Crypto API. This is used by the qmgr which directly manages the permanent Security Database held in filestore local to the running M-Switch.

  In addition there is a command-line tool which is used to load and manipulate information in this database. The use of this command line tool icmanage, is described later in this chapter.
- The permanent database can also be configured by MConsole over SOM to the qmgr.

The advantage of the permanent database is that it provides long-term storage. One use for CMS is that a message can carry certificates which are stored for later use in verification or encryption, or a certificate revocation list, used in the later verification of certificates.

The ephemeral Security Database is used by M-Switch components which can be configured using (for example) PKCS#12 files to hold certificates. These short lived components read these files and populate the ephemeral Security Database to use until the component has completed its work and exits, at which point the ephemeral Security Database disappears.

The long term plan is for all use of the ephemeral Security Database ceases in favour of using the the permanent Security Database.

8.6.2 Using the Configuration

Generally use of the Security Database should be managed by MConsole and access transparently by components of M-Switch such as SMTP channels. However there may be a requirement to access the permanent Security Database using icmanage. This section describes how to use the icmanage to access the Security Database.

If using a permanent database, the application will specify the file used for the database. There may be a default filename for the application. Configured trust anchors can be limited to the context(s) in which they are to be used. It is assumed that the configuration of the details for certificate verification are common across the application.

When the application needs a private key or permanent secret key, this is referenced by they key object's URI.

8.6.3 Security Database Details

This section describes how the security database stores its data and how to access the database.

8.6.3.1 Command Line Tool

There is a command line tool which is used to manage the database. It is invoked using the command:

```
icmanage -f <dbfilename> [-p <passphrase>] [-c] [<command>]
```

The passphrase is given using one of the formats explained below. The -c flag enables the database to be configured. If a command on the command line is not specified, commands are read from standard input.

Commands

help <command>
gives help on that command, or a list of commands if <command> is omitted.

change <old-passphrase> <new-passphrase>
: Change the passphrase for the database. If there is no passphrase, use an empty string.

cert <filename>
: Load a certificate or certificates from a file. If the filename ends in '*.pem*', then the file should contain one or more certificates in PEM format. Otherwise the file is assumed to contain a single certificate in DER format.

delete <object>
: Remove the object from the database.

pkcs12 <filename> [<passphrase>]
: Load a PKCS#12 file. If the file contains a private key and associated certificate they are created with related URIs, so that the certificate's URI can be used to retrieve the corresponding private key. Self-signed certificates within the PKCS#12 file are set as trust-anchors for "`any`".

pkcs8 <filename> [<passphrase>]
: Load a private key from a file. The file should contain a PKCS#8 format private key, which contains information on the key type.

private <type> <filename>
: Load a private key from a file. The file should contain a binary encoded private key of the type indicated. Possible types are:

  rsa
  : RSA key

  dsa
  : DSA key

  ec
  : Elliptic Curve key

read <object>
: Read configuration data from the internal database. If a partial URI is used then all matching data items are returned.

search [-f] <searchstring>
: Search for certificates. The search string is the query part of a certificate search URI, the part after the '?'. `-f` gives more information.

  When searching certificates, you need to omit the "`icrypto cert:?`" portion of the URI, e.g.:

```
icmanage> search -f valid=20130809161951Z
#1:  icrypto:cert:softkey:1376578855-20129-0 Certificate
subject: cn=dsa,c=gb
issuer:  cn=subca2,c=ca
serial:  04
valid:   20130809161951Z to 20180808161951Z
```

set <object> [<value>]
: Set configuration data, or object attributes. The object ID should be a full URI. If the value is omitted, then the attribute or item is removed.

show [-f] <object>
: Show data for an object. If the object is a certificate, then the corresponding public key is also displayed. If there is a corresponding private key in the DB then this is also displayed. `-f` gives more information.

8.6.3.2 Cryptographic Object URIs

Data objects in the database and their attributes are referenced using URIs. The general format is:

```
"icrypto:" <type> ":" <provider> ":" [<id>] [ ":" <attribute>]
```

For instance:

```
icrypto:cert:softkey:0000000002:trust-anchor
```

which refers to the attribute of a certificate indicating if it is a trust anchor.

8.6.3.3 URI Type

The <type> field gives the type of the data object, and can be one of:

config
: A generic configuration item

cert
: An X.509 certificate

prikey
: The private key of an asymmetric key-pair

pubkey
: The public key of an asymmetric key-pair

seckey
: A symmetric, secret key

crl
: A certificate revocation list

tls
: TLS specific configuration information

Normally, a private key is associated with a certificate loaded at the same time, normally from a PKCS#12 file. As a result, the URI for the certificate can be used to refer to the private key when the latter is needed for signing.

8.6.3.4 Provider

The provider indicates with which underlying cryptographic provider the data object is associated. Currently only the provider softkey is supported, which uses OpenSSL.

The ID portion is specific to the provider.

8.6.3.5 Attributes

Objects have certain attributes which are intrinsic to them. It is also possible to add meta-data to objects. This is mainly done for certificates. In particular:

trust-anchor
: This is a string which indicates the circumstances in which the public key in the certificate is trusted - forming the start of a certificate path.

private-key
: The URI of the private key which is the pair of the public key in the certificate.

<type>-<N>
: The <type> is the short form of subject alt. name key (see below), and <*N*> is a number. This adds what acts as an additional subject alt. name for the certificate. This enables one to find the certificate using a local value, such as the DN of an entry in the local Directory server.

8.6.3.6 Certificate Search URIs

It is possible to specify a URI which will initiate a search within the database for matching certificates. The URI must omit the "`icrypto:cert:?`" portion of the URI which are prepended automatically. See the Command Line Tool section below for an example.

The search terms are introduced by "`?`", and the search terms are: a key; "`=`"; and then a value depending on the key. If there is more than one term, they can be separated by "`;`" or "`|`". If there is more than one term, then all terms must match.

The available keys are:

serial
: certificate serial number (hex for the value part of the DER encoding of the serial number)

issuer
: DN string value for the issuer of the certificate

subject
: DN string value for the subject of the certificate

path-to-name
: DN string value of the path to name

subject-keyid
: hex for the OCTET STRING value of the subject key ID

issuer-keyid
: hex for the OCTET STRING value of the issuer key ID

key-usage
: Key usage - can be hex bit bask, using OpenSSL bit assignments, or a name for a single bit, one of: `digitalSignature, contentCommitment, keyEncipherment, dataencipherment, keyAgreement, keyCertSign, cRLSign, enciphermentOnly, decipherOnly`

key-algid
: Key algorithm ID expressed as a numeric OID string

valid
: Either a UTCTime string or a GeneralizedTime string giving a date/time at which the certificate should be valid.

pkey-valid
: A date/time when the private key should be valid.

subject-alt-name
: Test for certificate containing a subject alt. name of the given type. For `othername` type, the value is the numeric OID string for the type. Otherwise the subject alt name type string, as in the following items:

rfc822name or mail
: subject alt name, a email address string

dNSName or dns
: subject alt name, a hostname string

x400Address or x400
: subject alt name, in RFC 2156 string format

directoryNameor dn
: subject alt name, a DN string

ediPartyName
: subject alt name, not currently supported

uniformResourceIdentifier or uri
: subject alt name, a string

iPAddress
: subject alt name, not currently supported for values

registeredId
: subject alt name, not currently supported for values

trust-anchor
: matches the trust-anchor metadata attribute for a certificate. A presented value of “any” matches any attribute value. An attribute value of “any” matches any presented value. Otherwise the presented value must be a substring of the attribute value.

Searching for subject alt. name values, will include both values in the certificate itself, and meta-data values.

All keys and values are matched without regard to case.

8.6.3.7 Specifying Passphrases

There are several ways of specifying passphrases for the database as a whole and for data files being used for import.

pass:<value>
: `<value>` is the passphrase

file:<name>
: `<name>` is the name of a file which contains the passphrase in the first line

env:<var>
: The passphrase is taken from the environment variable called `<var>`

stdin
: The passphrase is read from standard input with a prompt if it is a terminal device

8.6.3.8 URI Cache

Objects can be referred to by their URI, or by the cache number returned from a referencing command. The cache reference is `@#@` followed by the number. It is possible to add an attribute name to the cache reference for an object, e.g.

```
#5:trust-anchor
```

8.7 Signing and Encrypting Messages

You can configure M-Switch to sign (or sign and encrypt) messages. The most likely need for this is as a message passes through a MIXER gateway, as the changing of the content will obviously mean that any signature is lost.

In order to do this, you must first configure your security environment, as described in ICrypto Config.

8.7.1 Internet to X.400

Once you have done set up the Security Environment, messages going from Internet to X.400 will automatically be signed.

Further changes can be made by editing the M-Switch Config, or the Internet Message conversion config file used by the mimeshaper channel `mimemixer-shaper.xml`, as described in S/MIME Channel Configuration.

8.7.2 X.400 to Internet

S/MIME signatures can be generated so that only the content is signed, or so that headers and content are signed.

Content Only

To sign just the content, you need a content converter something like this:

```
<convert type="content" action="extract">
  <match name="Content-type">oid\.1\.2\.840\.113549\.1\.9\.16\.1\.6
  </match>
</convert>

<convert type="header" action="sign">
  <param name="SmimeAction">mixed-signed</param>
  <filter command="ppmixer:ipms2rfc">
    <param name="convert-notif-req" />
  </filter>
  <filter command="pplablib:labelinsert">
    <param name="xheader-format">X-Protective-Marking: SEC=%c
    </param>
    <param name="subject-format">Subject: %d [SEC=%c]</param>
  </filter>
</convert>
```

This results in a message that looks like this:

```
headers
multipart/signed
    multipart/mixed
        text/plain
        text/plain
    application/pkcs7-signature
```

Signing Headers and Content

To cause the headers to be signed also, you need a content converter instead. Something like this will do what you want:

```
<!-- Converter for generating Signed output from P2,P22 or P772 -->
<!-- generates a multipart/signed  and message/rfc822
     which includes the headers -->
<convert type="content" action="sign">
  <match name="Content-type">(p22?|oid.1.3.26.0.4406.0.4.1)</match>
  <param name="SmimeAction">mixed-signed</param>
</convert>
```

You still need a header converter to insert labels:

```
<convert type="header" action="convert" cost="1">
  <filter command="ppmixer:ipms2rfc">
      <param name="convert-notif-req"/>
  </filter>
  <!-- extract label from subject of message -->
    <filter command="pplablib:labelextract">
    <param name="regexp">(?i)(^subject:.*\[Classification:([^]]*)\])
    </param>
    <param name="replace">$2</param>
    <param name="security-policy">uk-demo</param>
    <param name="priority">3</param>
```

```
  </filter>
  <!-- various label insertions including subject -->
  <filter command="pplablib:labelinsert">
    <param name="xheader-format">X-Isode-Label: %l</param>
    <param name="subject-format">Subject:%d [%l]</param>
  </filter>
  <filter command="pplablib:labelinsert">
    <param name="xheader-format">X-X411: %b</param>
    <param name="need-policy-id"/>
  </filter>
  <filter command="pplablib:labelinsert">
  <param name="xheader-format">SIO-Label: %e</param>
  </filter>
</convert>
```

You also need to comment out this convertor in the 822 output node

```
<!--
  <convert type="content" action="wrap">
  <match name="Content-type">(p22?|oid.1.3.26.0.4406.0.4.1)</match>
  </convert>
-->
```

You should then end up with a message structured like this:

```
headers
multipart/signed
    message/rfc822
        headers
        multipart/mixed
            text/plain
            text/plain
            ...
    application/pkcs7-signature
```

Note that there are two sets of headers - which will diverge as the message transits the MTS.

Note that the two configurations above both result in clear S/MIME signatures.

By changing the SmimeAction param to “`signed`” (whether headers are being signed or not), an opaque signature will result, i.e. a message structured like this:

```
application/pkcs7-mime
```

8.8 Strong Authentication

8.8.1 Channel Configuration

The x400p1 channel can be configured to use strong authentication when performing P1 Bind operations.

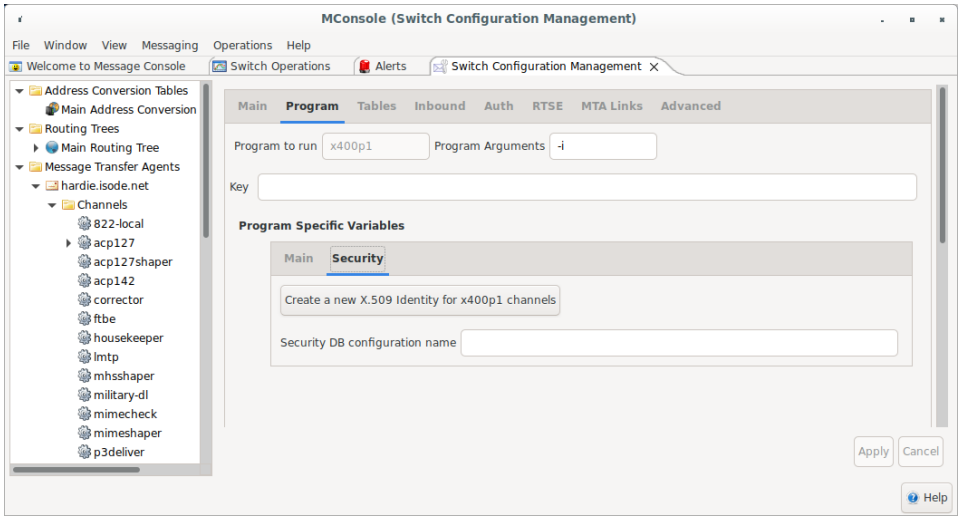
To configure strong authentication you should ensure the the **Authentication Requirements** for the Remote MTA P1 channel are configured to include **Strong Authentication**. **Simple**

Authentication and **MTA Name Present** can also be selected – the MTA will attempt strong authentication first if both strong and simple authentication are selected.

You can now configure the security environment for the X.400 P1 channel. This is shown in the diagram below, and described after that.

Note: The X.400 P1 security environment is now configured using the Security Database. From R19.1, the use of File Based security environment is no longer supported. See M-Switch Administration Guide.

Figure 8.1. Setting the X.400 P1 security environment.



Main Tab

Allow P1 Binds with Invalid X.509 Subject DNs
: The default of **No** means that Strong Binds using an X.509 Certificate whose Subject DN does not match the DN in the Bind, are not allowed.

Security Tab

Create a new X.509 Identity for x400p1 channels
: This button starts a wizard which creates a new PKCS#12 Certificate in the Security Database for use by x400p1 channels. This results in a Certificate Signing Request (CSR) being generated. This will require a Certificate Authority to generate the PKCS#12 for importing into the Security Database.

  The PKCS#12 is added into the Security Database with an index key which is a string used when an application such as the x400p1 channel wishes to reference the Certificate. By default this is the value **x400p1**. If no value is specified and the a Certificate with **x400p1** already exists, the Certificate is added into The Security Database with the value **x400p1_1 x400p1_2** ... as necessary.

Security Configuration Name
: The index key into the Security Database to be used by the x400p1 channel. If unset this is the value **x400p1**. If the index key **x400p1** is not in the Security Database the default **default** is used by the x400p1 channel.

If you check **AET Valid** in the **Authentication Requirements** then the X.400 P1 channel will not only ensure that the AET in the bind is valid (by reading the DN to retrieve the configuration of the remote MTA) but also check that the subject DN in the X.509 certificate provided in the bind matches the AET. You can disable the latter check by selecting **Allow Invalid DNs in Bind** on the **Main** tab.

8.8.2 Generating digital identities to be manually imported

To generate the cryptographic token required to enable Strong Authentication, the X.400 P1 channel must be provided with a Digital Identity. This must be in the form of a PKCS12 file including an X.509 certificate and private key.

Generally the wizard described in Section 8.8.1, “Channel Configuration” should be used to do this. However a PKCS#12 Certificate can be generated outside of MConsole and imported into the Security Database as described in M-Switch Administrator Guide

This is a three stage process:

1. Generating a Certification Signing Request
2. Generating and signing of the PKCS12 Certificate by a Certification Authority
3. Importing the PKCS#12 file into the security environment of the X.400 P1 channel

The Certification Authority may be a commercial organization such as Verisign, or you may use the Isode Certification Authority: SodiumCA.

8.8.2.1 Creating a CA Using Sodium CA

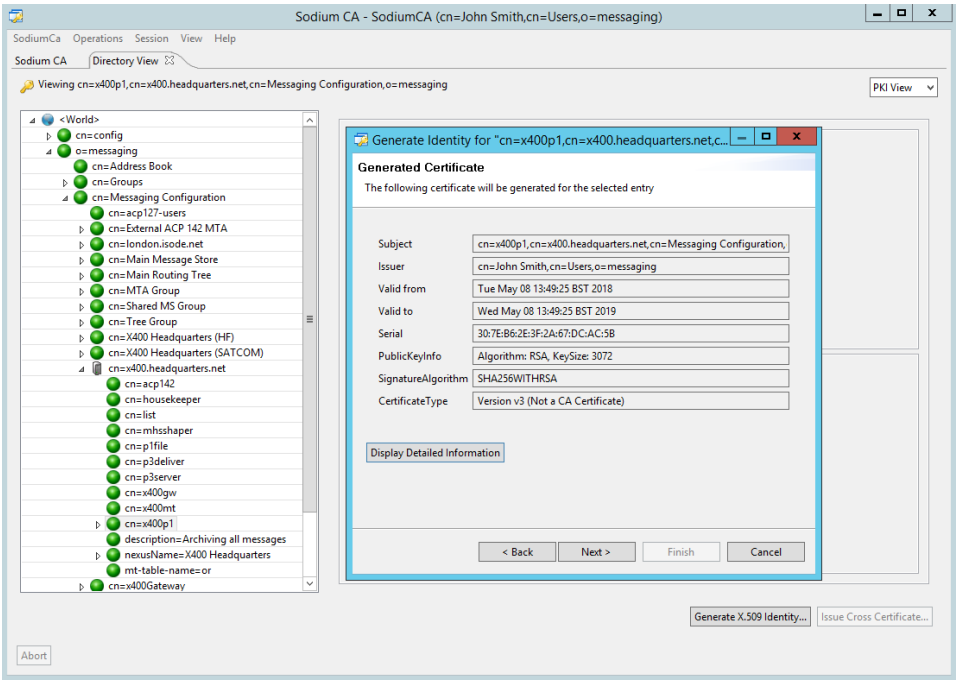
See the *M-Vault Administration Guide* to create the initial CA.

8.8.2.2 Generating a Digital Identity

This is done using the Isode tool SodiumCA. Run SodiumCA and browse to the **cn=x400p1** channel of the MTA for which you wish to generate the Digital Identity

Select the channel, and select **Generate X.509 Identity...**.

Figure 8.2. Generating A Certificate Signing Request: Selecting the Subject DN



You can now add a **Subject AltName**. For P1 entries, there are currently no values you are likely to want to add (possible future values are **Global Domain Identifier** and **MTA Name**).

8.8.2.3 Importing the PKCS#12 Certificate to the x400p1 security environment

Import the PKCS#12 Certificate generated outside of MConsole into the Security Database as described in M-Switch Operators Guide

Click on **Finish**.

You can now configure the passphrase for the private key. A random passphrase is generated for you or you can use your own value. In either case you will need to configure this value in a passphrase file in the directory in which the Digital Identity is being stored.

You need to ensure that you trust the certificate chain for the CA which issued your certificate. The CA which generated your Digital Identity should tell you what to do (if anything).

If you are using Sodium CA, you must ensure that each Digital Identity that is to be used by the local and remote X.400 P1 channels has the certificate of the issuing CA of both certificates in the security environment.

The CA certificate can be exported from Sodium CA into the Security Environment directory used by the P1 channel using the **CA Components** tab and exporting as DER. You should save the file as a *.crt* file in the directory configured to hold the trust anchors as described in M-Switch Administration Guide.

8.9 Table Based Authenticated Entities

Access to the M-Switch Queue Manager uses the Simple Authentication and Security Layer (SASL) to provide authentication for connections which use the SOM (Switch Operational Management) protocol. Usually the Directory is used to configure these Entities. This is described in M-Switch Administration Guide

For those Switches using table based configuration rather than the Directory, a simple XML table can be used for storage of SASL ids, passwords and associated access rights. This section describes how the Queue Manager is configured and makes use of these other resources using the XML table to configure the SASL IDs

To enable the use of XMLDB, add the following PP variables:

```
set sasl_auxprop_plugin=xmldb
set sasl_xmldb_file=”sasldb.xml”
```

Create the following file in *ETCDIR/switch/sasldb.xml*.

```
[?xml version="1.0" standalone="yes"?>
<sasl-data>
  <user name="John.Smith@isode.com">
    <att name="userPassword" servpass:encrypt="true">secret</att>
    <att name="somAccessRight">FULL</att>
  </user>
</sasl-data>
```

The standard `servpass` section allows the configuration of the Service Password service and verifier information. This allows SOM passwords elsewhere in the configuration file to be encrypted using the `spasscrypt` tool: these passwords are then decrypted at the

point of use. For more information on the Service Password mechanism, see M-Switch Administration Guide

Chapter 9 M-Switch Authorization

M-Switch is responsible for processing messages. This Chapter describes the post routing Authorization checks which can be configured to take place when the MTA is considering how to handle a message and its recipients.

9.1 M-Switch Authorization

The M-Switch Authorization features allows the configuration of rules which control the way in which messages are processed. For example:

- Controlling which routes for a message are permitted or blocked
- Controlling when a message is subject to content checking (e.g. for Viruses or Spam)
- Controlling when a message is subject to other checking (e.g. for Message size, or presence of Security Labels)
- Controlling when a message is archived
- Determining the inbound SMTP channel based on the calling SMTP system
- Controlling when a message is put into the *held* state on submission

There are two types of configurable items:

- **groups**: allow rules to use single reference to specify a set of entities, e.g. channels, IP addresses. Wildcards are supported.
- **rules**: a way of permitting or blocking messages and/or recipients.

A rule has the following components:

Description
: This is text which distinguishes the rule from other rules, and is useful for describing the rule's intent. When using the Corrector Channel, the text is used to help the operator identify the error. See M-Switch Administration Guide. In some cases the text is also used in logging and other places when the rule is applied.

Type
: This indicates the action to be taken when the rule is triggered

Priority
: When different rules are triggered, the rule with the highest numerical priority takes precedence. E.g. A rule of Priority 2 takes precedence over a Rule of Priority 0. This defaults in the UI to zero priority.

Filter
: The filter is a logical test on the properties of a message and recipient (or the SMTP caller for SMTP channel determination). If there is no filter, or the filter evaluates to `true`, then the rule is said to be triggered. Otherwise the rule is ignored.

9.1.1 Rule Types

Block, Permit
: These rule types work together to control if a message is allowed through or not. If the filter is `true` for a block rule and a permit rule with the same priority, then the block rule "wins" and the message is not permitted. These rules can be used for outbound channel selection based on different message and routing properties. They can also be used for other authorization purposes.

InChan
Used for inbound SMTP channel selection. Each such rule has an associated channel name. If the supplied channel key matches the key for the channel, and the filter is `true`, then that channel is selected. If no channel is selected, and there is a channel whose key value matches the key from the server and which does not have an associated InChan rule, then that channel is used as the default channel.

Archive
If the filter is `true` then the message type detailed in the subtype is archived to disk.

If the subtype is `Inbound messages`, only inbound messages will be archived to disk.

If the subtype is `Outbound messages`, only outbound messages will be archived to disk.

If the subtype is `Inbound messages and generate index`, only inbound messages will be archived to disk. It uses the content scanning subsystem, loading an appropriate indexer (see submitscanconfig.xml). The generated index is just a CRLF separated list of the "words" in the ia5text and subject of the message.

Archive by email
An email address has to be associated with the rule (either an RFC822 address or a string-encoded O/R Address). If the filter is `true` then the address is added to the message as an additional recipient (i.e. an "archive by email" recipient) and the message is not archived to disk.

Check
If the filter is `true` then the message is checked. If there is no associated channel list, then a checking channel is found which supports the content type of the message. If a channel list is associated with the rule, then that gives a sequence of checking channels to use.

Hold
If the filter is `true` then the recipients of the message are set to the held state. This means that no further processing of the message will occur until either the message is manually released (i.e. recipients set back to their normal state, allowing processing to continue), manually non-delivered or times out.

9.2 Authorization Groups

Authorization Groups provide a means for identifying set of entities which belong together and can be referenced collectively in Authorization Rules.

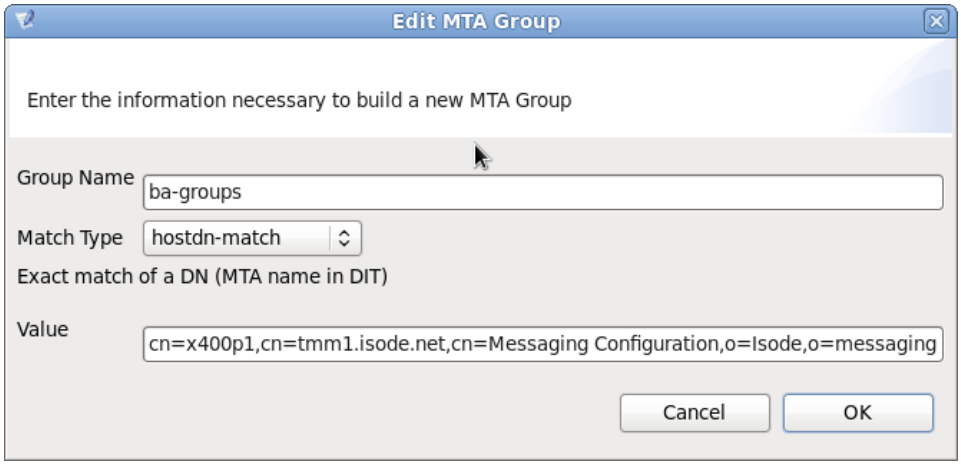
Groups: Can specify a set of

- Internet Address
- X.400 O/R Addresses
- Internet MTA names
- X.400 MTA Names (ie the DN)
- TCP Addresses (IPV4 or IPV6)

9.2.1 Creating An Authorization Group

To create an Authorization Group, Select MTA -> Authorization -> Add and enter suitable values into the Popup.

Figure 9.1. Creating an Authorization Group



Group Name

Enter the name of Authorization which wil be used in the Authorization Rule to refer to this Group.

Match Type

Enter the type of match to be used when comparing the the Value passed into the Rule during checking, with the value in the Group.

Possible Match Types are:

- email-match
- email-suffix
- email-pattern
- string-match
- string-suffix
- string-pattern
- oraddress-match
- oraddress-suffix
- oraddress-pattern
- hostname-match
- hostname-suffix
- hostname-pattern
- hostdn-match
- hostdn-suffix
- hostdn-pattern
- ipv4-match
- ipv4-pattern
- ipv6-match
- ipv6-pattern

Value

The value against which the Value passed into the Rule is checked. NB only a single value is permitted here which can include regular expression values if the Match Type is one of the "-pattern" types. To configure multiple Values when wildcards are not appropriate, each value requires a new Group, but with the group name duplicated.

9.2.2 Authorisation: Testing Group Entries and Matching

A command line utility is available to test if a value is in a group: `C:\Program Files\Isode\bin\grouptest`.

The usage for this is :

```
C:\Program Files\Isode\bin\grouptest <tag> <datatype> <value>
[<groupname>]
```

Example invocations:

```
C:\Program Files\Isode\bin\grouptest -help
```

```
C:\Program Files\Isode\bin\grouptest testTag d
"cn=x400p1,cn=tmm1.isode.net,cn=Messaging
Configuration,o=Isode,o=messaging"
```

Output:

```
# grouptest testTag d "cn=x400p1,cn=tmm1.isode.net,cn=Messaging Configuration
testTag: groups: ba-groups
```

NB the `tag` is an arbitrary value which is simply repeated in the output, but has no other function.

The above output shows that the DN is in the group ba-groups

9.3 Rule Filters

A rule filter is used to express the conditions under which a rule is triggered. It is a logical expression built up of basic components, combined by logical **and**, **or** and **not** operators. The basic components are tests on the properties of a message, a recipient of a message, or the SMTP caller for the case of a SMTP channel selection rule.

9.3.1 Basic Components

These have the form:

```
( item operator value )
```

for instance:

```
(inchan=smtp-external)
```

The type of the value depends upon the item and the operator. Items with values which are IP addresses, MTA names and Email addresses can be members of M-Switch Groups.

Within a filter special characters can be escaped by use of "\".

9.3.2 Building Filters

Filters can be built up from basic components combined with compound operators. Below, a *filter* is either a basic component or one of the constructs below, and a *filter-list* is the concatenation of one or more filters.

Negation Operator Formed by:

```
(! filter )
```

Negates the sense of the enclosed filter.

Boolean AND Operator Formed by:

```
(& filter-list )
```

Gives `true` if all the filters in the list give `true`.

Boolean OR Operator Formed by**:**

```
(| filter-list )
```

Gives `true` if at least one of the filters in the list give `true`.

9.3.3 Items

9.3.3.1 SMTP Caller Items

These are the only items which should be used in the SMTP channel rules (INCHAN rules).

inmta
The hostname of the SMTP caller, found by reverse lookup of the calling IP address.

inip
The IP address of the SMTP caller.

authid
The authorization ID of the message (SMTP AUTHID), the empty string if not authorized.

9.3.3.2 Standard Message Items

Note that, for the inmta and outmta items, the correct form for the MTA name is that which appears in ckadr output. For non-SMTP using directory routing this is the DN of the AE, not that of the AP.

If you are using mta in auth rules, the best way of doing this is to use a group and a suitably defined group member entry.

For authorization checks to be made the -i -m -s options and values must be specified.

inchan
The Switch channel on which the message arrived

inmta
The MTA from which the message was transferred

sender
The address of the sender of the message

authid
The authorization ID of the message (SMTP AUTHID) the empty string if not authorized.

recip
The recipient address

outmta
The (proposed) outbound MTA

outchan
The (proposed) outbound channel

type
The message type, one of message, report, probe

subtype
The message subtype. For X.400 messages, an Interpersonal Notification has a subtype of `ipn`, while a standard Interpersonal Message does not have a subtype value set. For Internet messages, the subtype field will be set for MIME messages of type multipart/report, from the `report-type` qualifier, so Message Disposition Notifications will match a filter of `subtype=disposition-notification` and Delivery Status Notifications will match a filter of `subtype=delivery-status`. Other Internet messages do not have a subtype set.

content
The content type of the message, the value being the standard M-Switch string representation.

size
The size of the message in bytes

priority
The message priority

The message priority is one of these values, ordered from high to low:

- override
- flash or urgent
- immediate
- priority or normal
- routine
- deferred or non-urgent
- bulk
- junk

9.3.3.3 Message Security Items

These rule items cause various security items to be checked. The clearance items use any security label in the message, and check that against the corresponding clearance using the configured security policy with the Access Control Decision Function. Further configuration is required for these items.

- **signature** Check any signature on the message. The possible values are (in increasing order): **fail**, **warning**, **absent** (i.e. no signature to check), **ok**
- **userclearance** Check label against recipient's clearance
- **mtaclearance** Check label against outbound MTA’s clearance
- **chanclearance** Check label against outbound channel's clearance. The possible clearance values are, in increasing order: **denied**, **absent**, **granted**
- **numsics** The number of Subject Information Codes in the message.

- **classification** The classification of the message. The possible clearance values are **no-label**, **unmarked**, **unclassified**, **restricted**, **confidential**, **secret** or **top-secret**. An integer value in the range 0 (no-label) to 6 (top-secret) can also be used.
- **label-errors** The number of errors which occurred during Security Label processing.

9.3.4 Operators

The available operators are:

=
: Gives true if the item matches the value. The value is an appropriate literal value.

:
: Gives true if the item is in the named group. The value is the name of a group.

+
: Gives true if the items have a group in common. The value is another item.

< <= > >=
: Comparison operators. Valid for items whose values are ordered. The value is a suitable value.

The first three operators can be preceded by `!` which negates the sense of the operator.

9.3.5 Example Filter

```
(|(outchan=x400p1)(outchan=x400-ba))
```

This example is True if one of the following is True:

- The outchannel is x400p1
- The outchannel is x400-ba

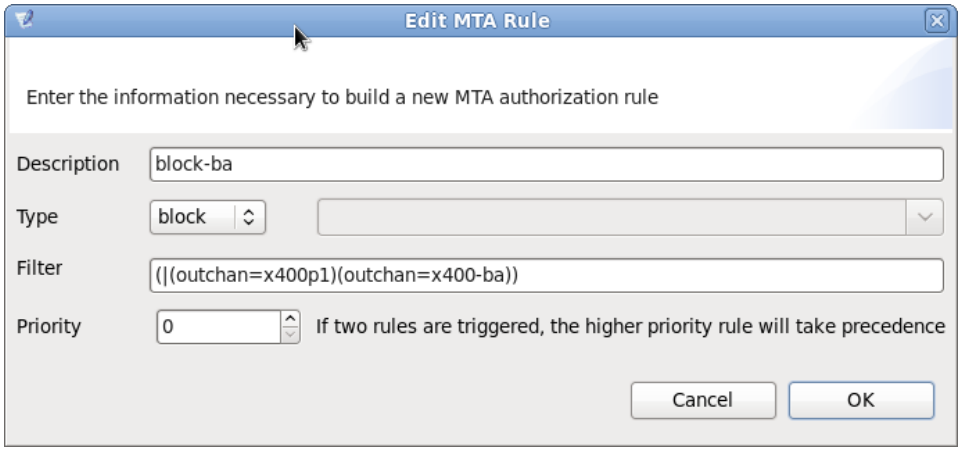
If Rule is a block Rule, that means message will be routed using this channel UNLESS a higher priority Permit Rule is present.

9.4 Authorization Rules

9.4.1 Block Rules

The following Rule is an example of a Block rule.

Figure 9.2. Example Block Rule

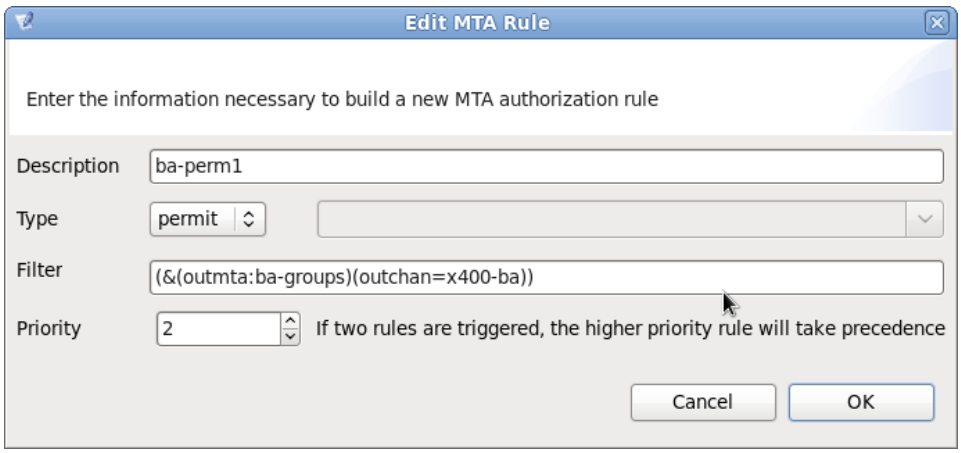


This Rule prevents and message being routed using the x400p1 channel or the x400-ba channel. The Priority is zero, which means that higher Priority rules can be used to allow the channels to be allowed under specific conditions (see Permit Rule section Section 9.4.2, “Permit Rules”).

9.4.2 Permit Rules

The following Rule is an example of a Permit rule.

Figure 9.3. Example Permit Rule



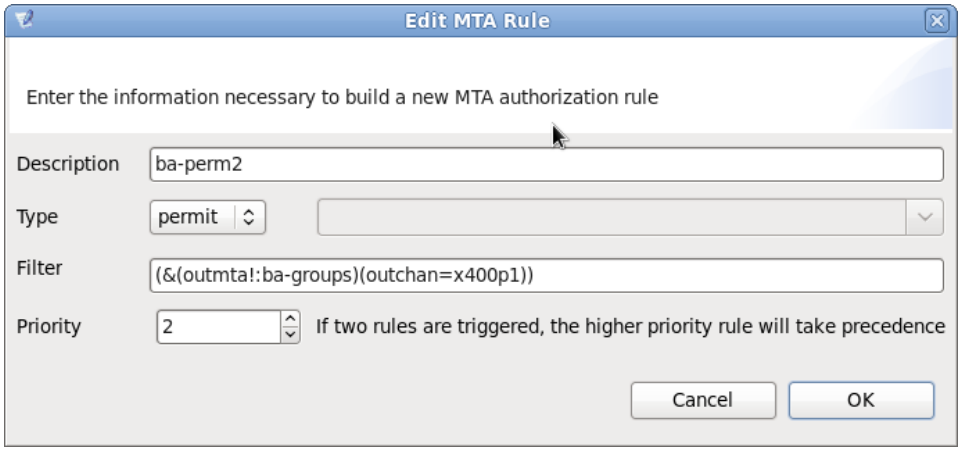
This Permit Rule has a Filter which is True if both the following are True:

- The outbound MTA is in ba-groups
- Outchan is x400-ba

I.e. this overrides the previous Block Rule so that messages to MTAs not in the ba-groups Group are transferred using the x400p1 channel.

The following Rule is a second example of a Permit rule.

Figure 9.4. Example Permit Rule 2



This Permit Rule has a Filter which is True if both the following are True:

- The outbound MTA is not in ba-groups
- Outchan is x400p1

this overrides the previous Block Rule so that messages to MTAs in the ba-groups Group are transferred using the x400-ba channel.

9.4.3 Combining Block Rules and Permit Rules

The above examples demonstrate an important principle often needed when configuring Authorization Rules. You need to combine Permit Rules at a higher priority than a block rule. The necessity of the Block Rule exists because configuring a Permit Rule on its own does not imply that a Permit rule bing False causes Routing to be blocked - rather a specific Block Rule must be configured.

9.5 Testing Authorization

Testing Authorization is carried out using the `ckadr` utility. By default `ckadr` is used to check Routing configuration prior to the application of Authorization Rules. This is described in multiple places in the M-Switch Manuals, including M-Switch Administration Guide.

9.5.1 Testing Authorization: ckadr Usage

This section describes `ckadr` invocation to report Routing results which include Authorization checks=.

```
usage: ckadr [options] [address...]

options:
    -a            Normalize all domains
    -d            DN submission
    -n            Not responsible (originator)
    -o            Set originator number
    -p (R|X)      Address parse only (as Internet or X.400)
    -r            Check as Internet address
    -v            Verbose mode
```

```
    -x            Check as X.400 address
    -z            Don't allow redirection
```

```
Authorization checking options:

    -c <type>     Content type
    -e <eits>     Encoded Info. Types
    -i <chan>     Inbound channel
    -l <size>     Message size
    -m <mta>      Inbound MTA
    -s <address>  Sender address
    -t (m|r|p)    Message/Report/Probe
    -u <prio>     Message priority
```

For authorisation checks to be made the -i -m -s options and values must be specified.

9.5.2 Testing Authorization: ckadr examples

The following shows that messages to MTAs not in ba-groups are transferred using x400p1.

```
# ckadr -v -x -s foo@example.com -i x400p1
    -m "cn=x400p1,cn=bevan.isode.net,
    cn=Messaging Configuration,o=Isode,o=messaging"
    "/CN=P7User1/O=tmm2/ADMD= /C=gb/"

/CN=P7User1/O=tmm2/ADMD= /C=gb/
    ->  (x400)  /CN=P7User1/O=tmm2/ADMD= /C=gb/
/CN=P7User1/O=tmm2/ADMD= /C=gb/
    -> (rfc822) "/CN=P7User1/O=tmm2/ADMD= /C=gb/"@bevan.isode.net

Delivered to cn=x400p1,cn=tmm2.isode.net,cn=Messaging Configuration,
    o=Isode,o=messaging by x400p1 (weight: 5)
Auth: S-MTA-AuthPermitted Route permitted: ba-perm2

Archiving
```

9.6 Closed User Groups

Groups are a mechanism which enables a Rule to refer to a set of entities with a single reference.

Groups are also used in conjunction with closed user group values to specify which entities can send to recipients which are within a closed user group.

A group is identified by a simple name, which can be anything. It is a good idea to use names which relate to the purpose of the group.

9.6.1 User Membership of Groups

Users (i.e. addresses) can be members of groups by the use of Directory attributes. The entry to which the attributes can be added depends upon the user type. For X.400 users, the entry is the “white pages” entry, unless there is no such entry, in which case the routing tree entry is used. For Internet users, it is the entry which is found using LASER routing lookup.

There are two attributes:

- **mhsUserGroup** for normal group members
- **mhsClosedUserGroup** for restricting senders to this address

The attributes are multi-valued, so the address can be a member of more than one group, and can be a normal group member, and a closed user group member. The auxiliary object class `mhsAuthUser` is used to enable the adding of these attributes to an entry.

Making the membership of a group or set of groups an attribute of a user entry in this way means that any Isode MTA which accesses the user entry in the course of routing or delivering a message can use the group membership(s) in Rule processing.

9.6.2 MTA-specific groups

The Directory attributes described in the preceding section are shared by all MTAs using that information. There can also be per-MTA group membership definitions. These can specify groups based on addresses and other types of value.

For a Directory-based configuration of the M-Switch system, the group membership information is held in the Directory, and then downloaded into an internal cached format.

For an M-Switch system which is not using the Directory to hold the information, the group information is defined in a text file, which is converted to the internal cached format using a command line tool.

The group membership information is a set of *triples*, each of which defines a way some values can be associated with a group. The three values are:

- **group name** The name of the group. You can have many items with the same group name
- **key type** The data type of the value.
- **key value** The value used in matching.

When a rule filter is checking if an item is a member of a group (or not) then only group membership rules of the appropriate data type are used. For instance, a sender address will only be checked against either Internet address items, or X.400 address items, depending on its value.

For a non-Directory based configuration, the text file is a sequence of lines, each of which has one *triple*, with the three values separated by colon characters. The table is built by running the program `groupbuild`. It takes one optional command line argument, which is the input file name. If this is not specified, then the input file is *grouptable.dat* in the configured table directory.

For a Directory-based configuration, group membership is configured in the Authorization tab for the MTA in question.

9.6.3 Special groups

There are a couple of groups for which there is automatic group membership.

empty The sender is a member of this group if it is absent, e.g. for an X.400 Report or an Internet DSN.

redirected A recipient is a member of this group if the address has been redirected locally.

9.6.4 Closed User Group Operation

If a recipient is a member of one or more closed user groups, by virtue of having one or more `mhsClosedUserGroup` attribute values, then the authorization system restricts the

senders who are allowed to send to that recipient. The sender must be a member of a normal group which has the same name as one of the closed user groups to which the recipient belongs.

The group membership of the sender can be derived by any of the group membership mechanisms.

To create a true closed user group, i.e. a set of users who can only send to each other, choose a unique group name, and set that name for both the `mhsUserGroup` and `mhsClosedUserGroup` attributes.

However, different kinds of group membership can be used to give access control to recipient addresses (e.g. mailing lists) based on other criteria. For instance, membership of the normal group corresponding to the closed user group could be specified as a partial address, which matches all internal addresses for an organization.

Chapter 10 Boundary MTA

Using M-Switch at the Boundary between Domains M-Switch has features which make it very suitable for use at the boundary between different domains.

10.1 Features

- Address Conversion
- Content Conversion. (See Chapter 6, *Content Conversion*).
- Modification of Acknowledgement Requests
- Generation of Acknowledgements at the Boundary
- Route dependent processing

10.2 Acknowledgements in a Boundary MTA

[This sections applies to R15.0 and later]

10.2.1 Acknowledgements

In this context, the term ‘Acknowledgement’ is used as a general term for these different messaging objects:

- X.400 Reports, both delivery reports and non-delivery reports
- X.400 Inter-Personal Notifications (IPNs)
- Internet Delivery Service Notifications (DSNs)
- Internet Message Disposition Notifications (MDNs)

Messages request these acknowledgements in different ways:

- Reports are requested through per-recipient user requests and mta requests
- IPNs are requested through per-recipient notification requests in the message heading
- DSNs are requested through a per-recipient SMTP extension (NOTIFY)
- MDNs are requested through a per-message heading field

When M-Switch acts as a MIXER gateway, converting between X.400 and Internet messages, it:

- Converts between X.400 Reports and Internet DSNs
- Converts between X.400 IPNs and Internet MDNs

In addition, the requests are converted, if possible:

- The X.400 user report requests are converted to/from the SMTP NOTIFY extension
- The IPN requests are converted to/from the `Disposition-notification-to` field in the Internet message heading

The latter in particular is subject to some configuration.

As a result of these equivalences, we will refer generically to:

- Reports, i.e. either X.400 Report or Internet DSN
- Receipts, i.e. either X.400 IPN or Internet MDN (this term is derived from the common usage for MDNs as ‘Read Receipts’)

Reference is made to report requests, and receipt requests generically.

When M-Switch generates a report or a receipt, the type depends upon the content type of the subject message. E.g. when generating a report, if the subject is X.400, then an X.400 Report is generated, if the subject is an Internet message, then an Internet DSN is generated.

10.2.2 Adding Acknowledgement Requests

M-Switch can change the acknowledgement requests to ensure the message requests reports and/or receipts. This would be used when it is known the target domain does generate acknowledgements, and you desire to ensure that it does even if the sender of the message had not requested them. Note that this can result in the sender receiving unexpected acknowledgements.

- To ensure that reports are requested, set the outbound channel's `boundary-ack` variable (set in the Advanced Tab of the channel's properties) to "`add-request`".
- To ensure the message content has a receipt request, set the channel's "`subtype out`" to the string "`add-ack-req`".

The latter will cause the relevant shaper channel to be used. The default configurations already have the converters for the outer heading which will set the required request. For the details, see the relevant content conversion filters.

10.2.3 Generation of Acknowledgements on Transfer or Delivery

M-Switch can generate reports and receipts when a message is successfully transferred or delivered. This feature is used when it is suspected or known that the receiving domain does not generate acknowledgements, and that it is desired to ensure that they are generated.

The feature is configured by setting the "`boundary-ack`" value for the outbound channel (configured in the **Advanced tab** of the channel's properties). These values are used for this feature:

gen-report
: Generate a report for the message's recipients, if requested by the message's originator.

force-report
: Generate a report, even if not requested by the originator.

gen-receipt
: Generate a report for the message's recipients, and a receipt for each recipient if requested by the originator. Note that is it possible in this case to get receipts but not a report.

force-receipt
: Generate a report and receipts, even if not requested.

It is also appropriate to suppress receipt requests within the content when using this option. If this is not done, the originator might be confused by multiple read receipts. To configure the outbound channel such that these requests are removed, set the channel's `subtype-out` to "`remove-ack-req`". This will cause to be used conversions with this effect, which are already set up in the standard shaper channel configuration XML files.

In a MIXER gateway, the type of the report or receipt is determined by the type of the message when it arrived, rather than by the type of the message when transferred or delivered, which can be different.

For an X.400 message, the mta requests control the optional generation of a report. If the originator has not requested the report, it will discarded rather than delivered to the originator. For the forcing cases, however, the user requests are altered so that the report will be delivered.

For an X.400 message, when receipts are not being forced, it is necessary to determine the originator's requests by matching the envelope recipient with the recipient specifier in the heading of the message, which is where the notification-requests are to be found. It is possible that there is no such match, e.g. following list expansion. In that case no IPN will be generated. This matches the correct behaviour of an X.400 UA.

For generated MDNs, there is a specific MDN type used for the generated text body and subject. These can be configured.

When these options are set, the envelope report requests are automatically changed, so that requests for positive reports are not sent to the receiving MTA. This suppresses the generation of positive reports later on. However, it is possible that a non-delivery report will be generated at some later point for the message. Any correlation of reports with messages should allow for this possibility.

10.2.4 Use of Multiple Outbound Channels

It is possible in a boundary MTA that you need to send messages to different domains with different characteristics. As the configuration of the features is controlled through the properties of the outbound channel, you may need to configure multiple outbound channels for sending to these different domains.

10.3 Use of Multiple Outbound Channels

There are occasions when it is necessary to have multiple channels of the same basic transfer protocol type (e.g. X.400 P1, or SMTP). The core reason is that is is necessary to have different values for items which are configured on a per-channel basis. For instance:

- content-out lists the content types which can be transferred over the channel.
- content subtype-out sets a subtype which controls aspects of content conversion.
- bodypart-out lists the X.400 exclusively acceptable encoded information types for the channel.
- protocol specific aspects of the way the channel connects to a peer MTA, e.g. the use of the `nomx` flag.

Chapter 11 Troubleshooting

Some of the troubleshooting tools and techniques described in this chapter can be followed routinely as preventative measures, as well as being used when a problem is encountered.

11.1 Checking the configuration

The **ckconfig** tool, which can be found in *(SBINDIR)*, performs various checks on the configuration of an MTA including:

- Checking that all the required directories are in place and have appropriate permissions.
- Checking that all the channels and filters, which are described in the MTA tailoring, exist and reference programs in *(LIBEXECDIR)*.
- Checking that any tables described in the MTA tailoring exist and reference files in the appropriate directory (as defined in the **Table directory** field on the **Advanced** page of the MTA's properties in MConsole).

ckconfig should be run on the system on which the MTA resides. Where there are several MTAs in a configuration, you should run **ckconfig** on each MTA host system. Although it is advisable to check the configuration with **ckconfig** before starting the MTA, the tool can equally be used while the MTA is running.

The command line for running this tool is:

```
ckconfig <options>
```

where options can be one or more of the following:

`-f`
: (force) **ckconfig** will attempt to automatically correct configuration errors. The default action is that **ckconfig** will prompt for correction of any configuration errors it identifies.

`-v`
: (verbose) **ckconfig** will display correct configuration information as well as displaying configuration errors.

`-n`
: (no execution) **ckconfig** will perform all its checks and report back the results. No automatic correction will take place and the user will not be prompted for corrections.

`-t <filename>`
: Use the file identified by filename as the *mtatailor* file instead of the default, *()mtatailor*.

It is worth running this program at intervals and after you make changes to the MTA configuration.

If you are not using Directory based routing and tailoring, you may also need to rebuild the MTA tables database after reconfiguration of the MTA. See M-Switch Administration Guide.

11.2 Checking addresses

The address checking tools, **ckadr** and **probe**, can be found in the directory *(SBINDIR)*. **ckadr** can be used with or without the MTA running.

Probe can only be used with X.400 addresses, and the messaging system needs to be running. In a configuration with several MTAs, the tools should be run on each system on which an MTA resides, to check the address handling configuration of that MTA.

11.2.1 ckadr

The **ckadr** tool checks whether an address is acceptable to the MTA. The tool can either be run from a command line or interactively. In both modes command options allow you to tailor its behaviour. The command line takes the form:

```
ckadr [<options>] <address> [<address>...]
```

where *<address>* is the address you want to check. If an address contains white space it should be quoted to avoid being split up into separate arguments. *<options>* can be one or more of the following:

`-a`
: Normalise all the domains specified in the address.

`-d`
: DN Submission

`-n`
: Not responsible (route as originator)

`-o`
: Set Originator number

`-r`
: Treat all addresses as RFC 822 format addresses. This is the default behaviour unless the system is configured as a pure X.400 system.

`-p`
: (R|X) Address parse only (as Internet or X.400)

`-v`
: (verbose) Extra information about the address is displayed, for example, redirection history.

`-x`
: Treat all addresses as X.400 format addresses.

`-z`
: Do not allow redirection

Authorization Options:

`-c <type>`
: Content type

`-e <eits>`
: Encoded Info types

`-i <inbound channel>`
: Sets all addressing characteristics based on the specified inbound channel

`-l <size>`
Message size

`-m <mta>`
Inbound MTA

`-s <address>`
Sender address

`-t <m|r|p>`
Message/Report/Probe

`-u <priority>`
Message Priority

In most cases, **ckadr** will correctly parse an X.400 address even if you say it is an RFC 822 address and vice versa. However, there are some cases where this is not so. You should therefore be careful when using **ckadr** and different address formats.

To run **ckadr** interactively, enter:

```
ckadr [<options>]
```

It will then attempt to read the address(es) to be checked from standard input, parsing each line of input as though it were an address. Any address containing white space should be quoted. To exit press the **Control** and **c** keys.

If **ckadr** is invoked with command line arguments other than those described above, it will attempt to parse these arguments as addresses.

No authorization checks are performed. An address may be acceptable to **ckadr** but the MTA may reject it later, when it is submitted for delivery, because of authorization restrictions. Configuring authorization is covered in Section 9.1, “M-Switch Authorization”.

If the address is acceptable, **ckadr** will display how the address was parsed and how the MTA will route this address. For example, the command line:

```
ckadr mjf@widget.com ir@widget.com
```

might return:

```
mjf@widget.com -> (rfc822) J.Ford@widget.com
mjf@widget.com -> (X.400) /I=J/S=Ford/PRMD=WIDGET/ADMD=GOLD 400/C=GB/

Delivered to sales.widget.com by 822-local

ir@widget.com -> (rfc822) I.Ritchie@widget.com
ir@isode.com -> (X.400) /I=I/S=Ritchie/PRMD=WIDGET/ADMD=GOLD 400/C=GB/

Delivered to sales.widget.com by 822-local
```

The first lines for both the above addresses show that the addresses `mjf@widget.com` and `ir@widget.com` have the aliases (or synonyms) `J.Ford@widget.com` and `I.Ritchie@widget.com` respectively. If you do not want **ckadr** to expand aliases, then include the `-n` option.

The second line shows the X.400 address that would be tried if the MTA was unable to locate the recipient using RFC 822.

The third line shows the MTA to which the message would be sent, `sales.widget.com`,and the channel it would use, `822-local`.

In a MIXER system, the following test could be used to check that incoming X.400 messages to this recipient would be delivered as RFC 822 messages:

```
ckadr -x "I=j; S=ford; P=widget; A=gold 400; C=gb"
```

A successful parse of the address would then be returned as:

```
I=j; S=ford; P=widget; A=gold 400; C=gb -> (X.400) /I=j/S=ford/PRMD
=widget/ADMD=gold 400/C=gb/
I=j; S=ford; P=widget; A=gold 400; C=gb -> (rfc822)
j.ford@widget.com
Delivered to sales.widget.com by 822-local
```

If the address is not acceptable, **ckadr** will display how far it went in parsing the address, and why it failed to find the address acceptable. For example:

```
ckadr -x "I=j; S=fort; P=widget; A=gold 400; C=gb"
```

might return the result:

```
Address parsing failed:
Reason: Unknown local user 'j.fort'
Parsing gave this:
I=j; S=fort; P=widget; A=gold 400; C=gb ->
(X.400) /I=j/S=fort/PRMD=widget/ADMD=gold 400/C=gb/
I=j; S=fort; P=widget; A=gold 400; C=gb -> (rfc822)
j.fort@widget.com
```

11.2.2 Probes

The **probe** tool checks whether or not a message would be acceptable either locally or to a remote mail system.

A message probe attempts to cross the Message Transfer System (MTS) network to its destination. On arrival at its destination, or at a place where it can no longer be delivered, a delivery report will always be returned. A message may be stopped before the required destination because of errors or because it has reached a gateway to a network which does not support probes. No message is ever delivered to a recipient as a result of a message probe.

The returned delivery report informs on the future successful or unsuccessful delivery of a message with the same parameters as those specified in the message probe. The report can be generated locally or at the remote end, depending on how the channels are configured in the MTA.

11.2.2.1 Configuring X.400 channels to support probe messages

There is no option to configure this. Channels either support probes or not. X.400 channels all support probes.

11.2.2.2 Running the probe tool

probe can be called with the command line options described below or interactively. With command line options, it will attempt to parse the specified options before generating a message probe. In interactive mode, it will prompt for the options on standard output and read on standard input until all the information has been received, before generating a message probe.

The command line takes the form:

```
probe -t <recipients> <options>
```

The options are:

`-t <recipients>`
probe interprets the arguments up to the next switch as `To: <recipients>`.

`-s <n>`
This checks if a message of size n bytes can be delivered to the recipient.

`-e <body types>`
This checks if the specified body part type(s) can be delivered. The list of types should be comma separated.

`-i`
Prohibits implicit conversion.

`-a`
Allows alternate recipients.

`-u <UA identifier>`
The UA identifier is a printable string, up to a maximum of 16 characters, which identifies the probe in subsequent delivery reports.

Note: The **probe** utility cannot be used in a pure X.400 MTA.

11.2.2.3 Configuring the MTA to reject probe messages

The `discard probes` option in the MTA tailoring allows you to configure an MTA to reject probe messages. In the MTA **Properties**/**Advanced** window of MConsole, select the discard probes option and tick the box.

11.3 Other checking tools

There are a number of small utility programs provided with the MTA that may be of interest to end users. These are described here.

11.3.1 A /bin/mail replacement

In *(BINDIR)/mail* there is a very simple user interface for submitting messages. It has approximately the functionality normally assumed by */bin/mail*, except that reading mail is not supported, only submission. That is, it takes a list of addresses on the command line and reads the body of the message on the standard input. By default all the addresses are marked as To: recipients. The command has a few options; each option stays in force until overridden.

`-f`
The following argument gives the `From:` address.

`-t`
The following arguments, up to the next flag, are `To:` addresses.

`-c`
The following arguments are `Cc:` addresses.

`-s`
The following arguments are text for the `Subject:` line.

This program is most useful in scripts and programs as a simple way of sending a message.

11.3.2 A /bin/sendmail replacement

In *`(EXECDIR)`/sendmail* there is a very simple user interface for submitting messages. It has approximately the functionality normally assumed by */bin/sendmail* on unix systems, except that only message submission is supported.

11.3.3 Messaging Configuration Integrity Checking

The *`(BINDIR)`/IntegrityCheck* script allows a set of integrity checks to be performed on an X.400 Messaging Configuration to detect inconsistencies which may have arisen in the course of configuration modifications. The following checks are performed:

- Scan all routedUA entries in the Routing Trees.

  If the routedUA entry represents a P7 Message Store user:

  - Check for the presence of an mhsMessageStoreDN attribute.
  - Check that the referenced icMessageStore entry exists
  - Check that the icMessageStore entry has an mhsORAddress attribute which corresponds to the routedUA entry.

  For all types of routedUA entry, if the routedUA has a White Pages DN:

  - Check that the referenced entry exists.
  - Check that the referenced entry has an mhsORAddresses attribute with a values which corresponds to the routedUA entry.
- Scan all White Pages entries, looking for entries which do not have an mhsORAddresses attribute value, or for any with a value which does not correspond to a configured routedUA entry.
- Scan all icMessageStore entries, looking for any "orphan" entry which does not have a routedUA pointing to it.
- Scan all Distribution List entries. For each Distribution List, obtain the set of list members (ORNames). If the ORName only contains a Distinguished Name, check that the entry given by the DN exists in the Directory. Otherwise, use the Queue Manager's "ckadr" functionality to check that the ORAddress component of the ORName can be routed successfully. This check is intended to identify the situation where an ORAddress has become invalid (e.g. a local mailbox has been deleted), but the address has not been removed from one or more Distribution Lists.

The script takes a number of arguments, some of which are mandatory:

`-w <password>`
Specifies the password to use to decrypt the Bind Profile file

`-b <bind profile name>`
Specifies the Bind Profile to use

`-c <configuration DN>`
Specifies the Messaging Configuration to check

`-l`
List the available the Bind Profiles

`-v`
Enable verbose logging

`-h <queue manager hostname>`
Hostname of Queue Manager to contact for routing check (defaults to localhost)

`-P <port>`
Port on which to contact Queue Manager for routing check (defaults to 18001)

`-u <userid>`
The SASL Id to use when connecting to the Queue Manager

`-p <password>`
The password to use when connecting to the Queue Manager

`-m <mechanism name>`
The SASL mechanism to use when connecting to the Queue Manager

An example of running the command is shown below:

```
IntegrityCheck -b localhost -c "cn=Mixer Messaging Configuration,
o=messaging" -w secret -u mtaadmin@badger.isode.net -p secret

Warning: White Pages entry <cn=TESTUSER,cn=White Pages,
o=messaging> has no mhsORAddresses attribute value of
/CN=TESTUSER3/O=isode limited/PRMD=ISODE/ADMD= /C=GB/

Error: routedUA <mHSCommonName=newtest2,
mHSOrganizationalUnitName=sales,mHSOrganizationName=isode limited,
pRMDName=ISODE,aDMDName=\ ,c=GB, cn=Main Routing Tree,
cn=Mixer Messaging Configuration,o=messaging> has invalid White
Pages DN <cn=newtest2,cn=White Pages,o=messaging>

Warning: White Pages entry <cn=David Wilson,cn=White Pages,
o=messaging> has no mhsORAddresses attribute value

Warning: White Pages entry <cn=newp3,cn=White Pages,
o=messaging> has no mhsORAddresses attribute value

Error: White Pages entry <cn=test2,cn=White Pages,
o=messaging> has an mhsORAddresses attribute value of
/CN=bogus/C=dk/ which does not exist as a RoutedUA

Error: White Pages entry <cn=zzzz,cn=White Pages,
o=messaging> has an mhsORAddresses attribute value of
/S=zzzz/O=Isode Limited/PRMD=ISODE/ADMD= /C=GB/
which does not exist as a RoutedUA
```

11.4 X.400 connection troubleshooting

Configuration of X.400 connections between MTAs using P1 is documented in M-Switch Administration Guide.

Connection failure or success will be recorded in the MTA logs. The event logs records provide information concerning the channel's view of the problem. The Audit log contains complete, if rather terse, information on the nature of the error and details of the connection attempt made. Explanation of the values and how to change the logging is provided in M-Switch Administration Guide.

If the connection fails, the following reasons may be given:

`Validation Failure`
: The remote site has either got an erroneous MTA entry in their tables, or they have no entry for your MTA.

`Unacceptable dialogue mode`
: This refers to the mode of RTS connection. Either the remote site does not support monologue or the RTS ASN.1 encoding is not working.

`Connect request refused on this network connection`
: The remote site is either down or its network listener is not running.

`Busy`
: The remote site is busy, try again later.

`Protocol Error`
: Protocol problems.

`Remote system problem`
: The process at the remote site has terminated abruptly.

`Timer expired`
: The process at the remote system is looping.

Connections are normally attempted when messages are queued for a remote X,.400 MTA. Alternatively you can cause a connection to be attempted for test purposes using the **ping** facility.

There are three ways you can do this:

- using the **x400p1** command with the `-p` switch (see below)
- using MConsole, right-clicking on the remote x400p1 channel and selecting **Test Connection**. See Section 11.5, “Testing remote X.400 connections with MConsole”.
- using the MConsole feature of opening a connection to a permanent MTA by right clicking on a permanent MTA. Permanent MTAs are present in MConsole for any MTA with whom the MTA has a peer connection configured. See M-Switch Administration Guide.

If the ping was successful, then try transmitting a proper message using an X.400 User Agent, such as XUXA, or the example X.400 API programs supplied with the product. The binaries are in *`$(BINDIR)`* and the source code is in *`$(SHAREDIR)`*`/x400sdk/example`. If during transmission a protocol error is received then this needs to be resolved. To analyse this problem, it may be necessary to obtain the required X.400 information by turning up the RTS, session, and transport logging. How to do this using MConsole is described in M-Switch Administration Guide. If you are not using MConsole to set up the MTA configuration, follow the procedure described in the *M-Switch Advanced Administration Guide* to modify the logging configuration.

Don't forget to reset the logging levels after the problem has been investigated.

In addition, or alternatively, the x400p1 channel can be run in debug mode. In this mode the user has to act as the qmgr in the protocol exchanges. The form of the command line is:

```
x400p1 debug [-c <channel>]
```

where

`debug`
: Starts the channel in debug mode.

The other options are as described in the *M-Switch Advanced Administration Guide*. The channel would normally be started with a command line like:

```
x400p1 debug -c x400p1
```

You then need to interact with the channel acting as a proxy for the qmgr. In order to send a message, the following sequence interchange occurs:

```
{
   channelStart
}
{
   channel "x400p1",
   maxinst 1,
   appltype 2,
   flags 7,
   status 2000000
}
Operations:
        Message Success Timeout ProtocolError  AuthenticationError
        Congested Start Connect Disconnect Abort
        Accepted Rejected Pause Resume Closedown
Operation (unique prefix): success
Data:
Instance [1]:
Request [0]:
{
   channelRequest
}
{
   instance 1,
   request 1,
   timelimit -1,
   priolimit 8
}
Operations:
        Message Success Timeout ProtocolError  AuthenticationError
        Congested Start Connect Disconnect Abort
        Accepted Rejected Pause Resume Closedown
Operation (unique prefix): connect
Data: cn=x400p1,cn=attlee-sink1,cn=Messaging Configuration,ou=MHS,c=GB
Instance [1]:
Request [0]:
Pinging cn=x400p1,cn=attlee-sink1,cn=Messaging Configuration,ou=MHS,
c=GB
Connected Successfully to cn=x400p1,cn=attlee-sink1,cn=Messaging
Configuration,ou=MHS,c=GB
{
   channelStatus
}
{
   instance 1,
   request 2,
   context 1,
   status 2000000
}
Operations:
        Message Success Timeout ProtocolError  AuthenticationError
        Congested Start Connect Disconnect Abort
        Accepted Rejected Pause Resume Closedown
Operation (unique prefix): message
MessageID: msg.11965-0
Enter recipient number, one per line:
        > 1
        >
{
   messageStatus
```

```
}
{
   instance 1,
   request 1,
   rdone 1,
   sdone 839,
   recips {
      RecipInfo {
         rno 1,
         opstatus 1,
         status 0
      }
   },
   newrequest 3,
   timelimit -1,
   priolimit 7
}
Operations:
        Message Success Timeout ProtocolError  AuthenticationError
        Congested Start Connect Disconnect Abort
        Accepted Rejected Pause Resume Closedown
Operation (unique prefix): disconnect
Data:
Instance [1]:
Request [0]:
Operations:
        Message Success Timeout ProtocolError  AuthenticationError
        Congested Start Connect Disconnect Abort
        Accepted Rejected Pause Resume Closedown
Operation (unique prefix): closedown
Data:
Instance [1]:
Request [0]:
```

The commands entered by the operator are in bold. They are

```
success
connect
   data <mtaname>
message
   <msgid>
   <recipnum>
disconnnect
closedown
```

11.4.1 How to test an X.400 connection

Use the x400p1 channel in ping mode to attempt to connect to a remote MTA. The command line would take the form:

```
x400p1 ping|-p -a<addr>|-m<mtaname> [-c<channel>]
```

Either `-a` or `-m` should be used to specify the connection to be tried. The option meanings are as follows:

`-p`
: Start the channel in ping mode, which means attempt to connect to a remote MTA and exit, reporting on success or failure. ping or -p may be specified.

`-a<addr>`
: The value specified should be an OR-address. The channel connects to the MTA to which a message addressed to this recipient would be relayed.

`-c<channel>`
: Claim to be the given channel name, <channel>, on input. If this option is not set, the channel used is that found by a lookup of the program name as a channel.

`-m<mtaname>`
: If you have configured Directory based routing, this is the Distinguished Name of the MTA which is read to obtain the connection information. If you are using table based routing, it is the name used as key into the outbound channel table.

The following command line examples starts the x400p1 channel in ping mode to test that a remote MTA can be reached.

If the addressing information is held in the Directory, the channel can be started in ping mode specifying either the Distinguished Name of the MTA entry as the value of *<mtaname>*, as in

```
x400p1 -p -m"<cn=Remote MTA,ou=server,o=Myorg Corp,c=US>" -cx400p1
```

or the OR-address to be used for routing to the remote MTA as the value of *<addr>* in the -a option, as in the example

```
x400p1 -p -a "/I=D/S=Smith/O=Myorg/ADMD=Sprint/C=US/" -cx400p1
```

11.5 Testing remote X.400 connections with MConsole

The easiest way to check X.400 connections, is using the **Test Connection from this MTA** feature of MConsole.

To do this, select the remote MTA you want to test the connection to (the responder MTA), expand the **Channel** folder, and select the **x400p1** channel. Right click, and select **Test Connection from this MTA**. A window is displayed, giving you the choice of selecting the MTA you are going to test the connection from (the initiator MTA). In order to perform this test, you will need to contact the SOM daemon of the initiator MTA, so you'll need the user name and password. This is the same that you use for MConsole. After you have authenticated successfully, you can click the **Test** button at the bottom of the window.

Note: The M-Switch event log records the information reported below. The M-Switch audit logs also record all attempted connections.

Below are some common connection errors, their likely cause and how to fix them.

11.5.1 Cannot connect to remote MTA

This error message means that it wasn't possible to establish a connection to the remote MTA.

```
Connection from voshod.isode.net to sumo.isode.net failed: Command
failed (connect request refused on this network connection)
```

This could be because:

- The remote MTA is not running. If you can, check that the process is running normally, and if its a Isode MTA, that IAED is running. If it is not under your control, ask the administrator of the other MTA to check it for you.
- The remote MTA is running, but there's a firewall that prevents connecting to port 102 (the default OSI port).
- You have incorrectly configured the remote MTA Presentation Address

A simple way to establish that there is network connectivity is to use the **telnet** program. To check that you can connect from `host.example.com` to port 102 on `remote.example.com`. You can use:

```
telnet remote.example.com 102
```

If you get this message:

```
telnet: Unable to connect to remote host: Connection refused
```

then either IAED is not running on the remote host, or there is a firewall blocking access to port 102. In Linux, check that IP tables is correctly configured in the remote host. Similar checks applies for Windows firewalls.

If it works, you should see a message similar to this:

```
Connected to remote.example.com
```

11.5.2 Incoming X.400 Connections

The first thing you should check is that you have connectivity.

Start the Isode MTA software, check that logging is being written in *$LOGDIR* (*/var/isode/logs* or *C:\Isode\log*) and then ask the administrator of the remote MTA to run the **telnet** command:

```
telnet <your-IP-address> 102
```

If there is IP connectivity, no firewalls stopping connections to port 102, and your software is running and correctly configured, then the administrator of the remote MTA should see something like this:

```
Trying 192.168.0.20...
Connected to spec.isode.net.
Escape character is '^]'.
```

Ask the administrator to type the word "`test`" and then close the connection with **Ctrl-D**.

As the word "`test`" is an invalid command to issue, you should see this in the *mta-event.log* (or *iaed.log* in case you have split logging).

```
iaed 14015 (root) F-Tsap-BadITOTversion Bad ITOT version number 116
```

If this is not happening, then check that they can, for example, ping or ssh to your machine.

You may also want to check that *isode.iaed* is listening on the TSAP port (TSAP == 102). Try

```
netstat -a | grep tsap
```

you should see something like this:

```
tcp 0 0 spec.isode.net:iso-tsap 0.0.0.0:*   LISTEN
```

11.5.3 Authentication errors

The MTA that initiates the connection is being rejected by the responding MTA. This error message could mean two things:

```
Testing connection to sumo.isode.net....
Connection from voshod.isode.net to sumo.isode.net failed: Command
failed (validation failure)
```

- A connection to the remote MTA was established, but *they* rejected *our* authentication information.
- A connection to the remote MTA was established, but *we* rejected *their* authentication information.

To find out which one is the case, you can examine the mta-event log and mta-audit logs:

- If we reject their authentication information, the *mta-event* log will explain the error in more detail.
- If they reject our authentication information, the *mta-event* log will simply report `authentication error`.

11.5.4 Remote MTA not found in our configuration

A responding MTA relies on the MTA Name supplied in the P1 bind to look up the remote Initiator and check the authentication information. An incoming connection is rejected if it cannot find it in its configuration.

Check that the MTA that is trying to connect is correctly configured in your system. Then generate the MTA Links of your MTA's x400p1 channel in MConsole by right clicking on the **Message Transfer Agents** folder then selecting **Create MTA Links**.

11.6 Troubleshooting P3 connections

When you create an X.400 P7 Message Store user with MConsole, by default it sets the PP Channel to `p3deliver`. This is because the MTA will deliver the message to the Message Store via P3. As message is delivered, then its no longer on the Queue, its in the Message Store.

Now, if you don't want to use an X.400 P7 Message Store, what you do is to create X.400 P3 users in MConsole that have the PP Channel set to `p3server`. This means that when a message arrives to this user, it will be put in the `p3server` channel. You can then see it in the Queue using Mconsole. Until you connect to the `p3server` channel as the user, the message won't be delivered to you.

The XUXA X.400 User Agent program, when correctly configured to use X.400 P3, will read (i.e. deliver) all the messages that are ready to be delivered to the X.400 P3 user.

You should to note that our sample command line User Agent programs (both the Tcl and C sample receive programs) connect to read just one message.

This means that when they finished reading one message, they just quit. If the MTA has more than one message in the `p3server` channel for that user, it will try to deliver the second message, but the UAs are not prepared to receive it, and so the delivery of the second message fails. Because of this failure, the MTA puts a delay on the `p3server` channel. So that's why you connect again to read messages, and they are not delivered to your UA, even though they are on the queue.

If this happens, you can do three things:

1. Make sure that your P3 User Agent programs always check to see if there are more messages waiting before quitting or closing the p3server channel connection. (This is the preferred option, long time)
2. Clear the delay in the `p3server` channel with MConsole (with **Clear Delay** or **Downwards force attempt**). This is good for testing only.
3. Restart the server, as this will clear the delay. This is far too drastic, no need to do it this way.

If you open the connection to the `p3server` channel, the MTA will issue a message delivery operation. If you quit without reading the message, the delivery fill fail.

If you look at the Queue using MConsole, you will find that the message is not gone. This is because the MTA Delivery to the UA fails, the message is marked not available as it is delayed. So when you connect again, there's one message less available for reading. When the delay is cleared, it will be available again.

11.7 Basic Message Tracking

This section gives some tips to evaluators on what to do if you sent a message but cannot see it arrive.

1. In general, the best please to start is to look in the logs: (`/var/isode/log` or `C:\Isode\log`). The most likely files that will show useful information are `mta-event.log`and `xms-event.log`. If the message was indeed received it should be shown in `mta-audit.log` and `xms-audit.log`.
2. The message may have been sent, but not delivered, for example, because of the sample latest delivery time was exceeded. Check if you set the latest delivery time value in the example. In any case, the sender should have received a delivery report.
3. The message is stuck in the queue, for whatever reason. In this case, you'll find a directory called `msg.XXXXX-N` under `/var/isode/switch` (`C:\Isode\switch`). You can also see this message and under which channel it is with the MConsole tool (`/opt/isode/bin/mconsole` or the MConsole Windows shortcut).
4. Finally, the message may have been delivered, but you cannot see it. In this case, you'll see new messages being delivered with either MConsole X.400 Message Store Operations View, or by looking at the recipient user's mailbox, which is usually under `/var/isode/mailboxes/` or `C:\Isode\bin`.

If you still cannot find what happened to your message, send the log files, together with a short description of who you are sending from and to to `support@isode.com`.

Of course, if you have already set up the Message Audit Database, it would be much easier to track the message by using any of the MConsole views under Audit Information (**Message History**, **Message Tracking** and **Message Transfers History**).

11.8 Preventing messages from being deleted

You can prevent messages from being deleted from the M-Switch queue by setting the **No delete** option of the MTA. You can do this by using MConsole.

Select your local MTA in the **Switch Configurations View**, select the **Advanced tab**. In it you'll find the option **No delete**, with a check box that can be set.

Then the message will be in the queue */var/isode/switch* or `C:/Isode/switch` in a directory called something like `msg.12345-0`

Bear in mind that, as with other settings configured by MConsole, it may take some time for the QMGR to spot the change, and to write it in the `mtatailor.tai` file.

11.9 M-Switch X.400 Logging

To increase the logging generated when you test an X.400 connection from MConsole, for a you should do this:

1. In MConsole, expand your local MTA, and the **Logs** folder under it. Select the **Eventlog**, right click and choose **New Program-specific Stream**. Select **x400p1** and click on **OK**. Expand the **Eventlog** node.
2. Select the **x400p1** entry, enter *x400p1.log* in the **Log file Name** field, select the **Audit and Event logging** tab. In the **Events** section click on the **Advanced...** button. Select the **MTA_X400** row and click on the **Edit** button, and then check all the **Facility Levels** boxes except PDU, click on **OK** and then on **Apply**.
3. Restart the QMGR. On Windows: using Isode Service Manager, stop and start isode.pp.qmgr. On Unix, use the pp startup script (usually */etc/init.d/pp*) to stop and start the whole MTA.
4. Try testing the connection again, using the **Test Connection from this MTA** option on your MTA's `x400p1` channel.
5. You should see a new log file called *x400p1.log* under your log directory (*/var/isode/log* or *C:\Isode\log*).

11.10 Messaging System Checks

11.10.1 Overview

MConsole provides a way to run some checks that report on the DNS configuration of Internet domains, which can be your own domain or an external domain.

This tool can be useful to establish if the locally configured SMTP server has been set up to use all of the security settings that Isode M-Switch supports.

The tool is not able to modify your Isode M-Switch configuration or the DNS settings, but it can be useful in determining which features are configured and what are the DNS values.

It can also be useful to determine what is the configuration of an external MTA, by looking at the information available in DNS about a third-party domain.

11.10.2 Running the DNS System Check

To run the System Check, open the Switch Configuration View then select the menu **Messaging** → **System Check**.

To report on the configuration of a domain you have two options: you can select one of your local domains from the **Check local domains** combo, or you can manually enter a domain in the **Check this domain** box.

Once a domain has been provided (either selected from the Combo box or manually typed), the **Check** button will be enabled. To run the report, click on the **Check** button. The action will make MConsole perform a number of DNS searches and report the result in the HTML browser shown below.

The figure below shows a sample report generated for the domain `isode.com`.

There are currently five areas that are reported: MX Records, SPF, DKIM, STS and DMARC.

If a feature is configured in DNS, the information available will be shown. For example, in the image above, the SPF feature is configured, and the value is `v=spf1 mx ~all`.

If a feature is not configured in DNS, for example, there are no records for STS, then the report will include the line: `>>> Doesn't have STS Records`

M-Switch does use A records if the MX lookup for a domain returns no values. In this case, if the lookup of the domain as a A record is successful, the returned value is used to connect. The DNS system check feature does not currently check the A records for the domain.

11.10.3 References

- *MX*

  Summary: The MX (mail exchange) records

  RFC Reference: RFC 1912

  RFC Link: https://tools.ietf.org/html/rfc1912

  Example: `isode.com. 86400 IN MX 1 waldorf.isode.com.`
- *SPF*

  Summary: Sender Policy Framework (SPF) for Authorizing Use of Domains in Email

  RFC Reference: RFC 7208

  RFC Link: https://datatracker.ietf.org/doc/rfc7208/

  Example: `v=spf1 include:_spf.google.com ~all`
- *DKIM*

  Summary: DomainKeys Identified Mail (DKIM) Signatures

RFC Reference: RFC 6376

RFC Link: https://datatracker.ietf.org/doc/rfc6376/

- *STS*

  Summary: SMTP MTA Strict Transport Security / SMTP TLS Reporting

  RFC Reference: RFC 8460, RFC 8461

  RFC Link 1: https://datatracker.ietf.org/doc/rfc8461/

  RFC Link 2: https://datatracker.ietf.org/doc/rfc8460/

  Example: `v=STSv1; id=20171114T070707;`

- *DMARC*

  Summary: Domain-based Message Authentication, Reporting, and Conformance

  RFC Reference: RFC 7489

  RFC Link: https://datatracker.ietf.org/doc/rfc7489/

  Example: `v=DMARC1; p=reject; rua=mailto:mailauth-reports@google.com`

Chapter 12 Tips

This sections provides tips on how to configure M-Switch in unusual ways.

12.1 Installing the software on non-standard paths

On Windows, it is possible to select the path where the software is installed by changing the default value suggested by the Windows installer.

On Unix, the easiest way to make the servers use non-standard paths is to use symbolic links. So install the packages and accept the default locations.

Say that you want to use */mnt/isode* instead of */var/isode* for all the usual directories. Then you should:

- Stop all the services (if they are already running)

```
service pumice stop; service pp stop ; service dsa stop
```

- Copy the existing directories

```
cp -pr /var/isode/* /mnt/isode *
```

- Move the */var/isode* directory out of the way

```
mv /var/isode /var/isode-old
```

- Set up the symbolic links

```
ln -s /mnt/isode/archive /var/isode/archive
ln -s /mnt/isode/dsa-db /var/isode/dsa-db
ln -s /mnt/isode/log /var/isode/log
ln -s /mnt/isode/mailboxes /var/isode/mailboxes
ln -s /mnt/isode/switch /var/isode/switch
ln -s /mnt/isode/tmp /var/isode/tmp
```

- Make sure that everything is OK, and then start the services

```
service dsa start; service pp start; service pumice start
```

- When everything is working fine, remove the /var/isode-old directory

```
rm -fr /var/isode-old
```

12.2 How can set a limit on the size of a message?

There are several ways to set a limit to a message size.

12.2.1 Setting a per-channel maximum message size

You can configure a maximum message size by using the MTA Authorization mechanism. You can impose a size limit on a channel, (let's say `p3deliver` or x400p1), and all messages that exceed that size will get a DR (Delivery Report).

In MConsole, edit your local MTA and select the **Authorization** tab. To add a new Rule, click on the **Add** button. Give the new Rule a description, for example "`P3 message size limit`". Select the **Type** as **block** and in the **Filter** value enter, for example:

```
(&(size>1000000)(outchan=p3server))
```

Click on **OK** and then **Apply**.

12.2.2 Setting a per-user maximum message size

You can also set the size on a per user basis, again using authorization, but even though this is very flexible, it requires you to keep it up to date manually, that is, after creating a user with MConsole, you need to enter it's authorization entry in the table (again using MConsole).

12.2.3 Preventing the submission of large messages

The other option would be to limit the submission of the message from the UA size. The X.400 Demo User Agent (XUXA) uses a Directory based Address Book, and one of the optional attributes that users in the DSA can have is `mhsDeliverableContentLength`. If that attribute is present, and the message size the UA tries to submit is bigger, it won't do it and inform the user.

Of course, this is only half of the story, as messages can come to the MTA via say P1. You can put a limit there (using channel authorization).

In the future we will make the MTA check the user's DSA entry, and read the `mhsDeliverableContentLength`, but this work hasn't been planned yet.

12.3 Use of iaed in Table-based Configurations

Inbound X.400 P1 and P3 connections (for message transfer in or submission) are initially handled by iaed. This then starts the appropriate responder channel for the called Presentation Address (e.g. x400p1 for incoming X.400 P1 connections).

In a normal M-Switch installation, iaed obtains information about the set of Presentation Addresses to listen on, and the corresponding responder channels by searching the Directory. In a table-based configuration, iaed instead obtains this information from the

(ETCDIR)/isoservices file. In this case it is referred to as running in tsapd mode (this was the original name of a separate application which ran in table-only mode).

A suitable entry for X.400 P1 connections might be:

```
"tsap/p1" "591" /opt/isode/libexec/x400p1
```

In this example, the service name is `tsap/p1`, the transport selector is `"591"` and `/opt/isode/libexec/x400p1` indicates the location of the X.400 P1 channel program (and any arguments). If inbound X.400(84) messages are expected, use a numeric IA5 form for the transport selector.

Note: Normally, isode.iaed operates in "iaed mode" (i.e. using the Directory) when there is a `loc_mtadnname` key in the *mtatailor* file, and in "tsapd mode" when there is no such key present. You may override this behaviour, and force isode.iaed to operate in tsapd mode by starting it with the `-F` command line argument.

12.4 Non-standard Use of the X.400 Channel

The X.400 P1 channel can be run in a number of non-standard ways from the command line to perform specific functions, as described below.

12.4.1 Starting the X.400 Channel from the command line to pull messages

This option might be used to retrieve messages from a remote MTA that can only be contacted intermittently.

Note that the -i flag is only used when the program is invoked by the QMGR and not when run from the command line

The following options may be included on the command line, and will override any corresponding values specified in the channel configuration:

```
x400p1 -Ip -m<mtaname> [-te|d] [-r|-E] [-f<logname>] [-n]
      [-c<channel>]
```

`-Ip`
: Start as initiator to pull messages. This value is mandatory.

`-m <mtaname>`
: This value is mandatory. In a Directory based configuration, `mtaname` will be the Distinguished Name of the MTA which is read to obtain the connection information. In table based configurations, it is the name used as a key into the channel table.

`-t <suboption>`
: The value of `suboption` can be either `e`, to enable or `d`, to disable two way alternate mode on all connections. In table based configurations, if a value is given here it will override the value of the mode field in the channel table.

`-E`
Enable use of checkpointing and recovery facilities. If this option is specified, the channel will attempt to resume the transfer of a previously aborted message.

`-s`
Enable saving of checkpoint data as the message transfer proceeds. The `-s` flag provides additional protection in the case of a system crash, as the necessary checkpoint data will always be saved.

> **Caution:** Associations which fail while the initiator is receiving a message cannot be recovered.

`-f <logname>`
Send logging to the file specified by `logname`. By default, the program name `x400p1` is used.

`-n`
Use 1988 stack (default).

`-c <channel>`
Claim to be the channel, `channel`. By default the channel is assumed to be the name of the program itself.

The following command line might be used to start the X.400 channel in a Directory based configuration to pull messages:

```
x400p1 -Ip -c x400p1 -m "cn=garfield, o=widget ltd, c=gb"
```

`cn=garfield, o=widget ltd, c=gb` is the Distinguished Name of the MTA which is read to obtain the connection information.

The following example illustrates a command line for a table based configuration, where the channel table holds the information required to connect to the MTA called x400.headquarters.net:

```
x400p1 -Ip -c x400p1 -m x400.headquarters.net
```

12.4.2 Starting the X.400 channel from the command line to recover a failed message

> **Note:** This option is not currently supported.

The channel can be invoked from command line to recover from a failure which occurred when the initiator was receiving a message. The following command line options can be set:

```
x400p1 -Ir -m<mtaname> [-te|d] [-r|-s] [-f<logname>] [-n]
      [-c<channel>]
```

`-Ir`
Start as initiator to recover inbound failures. This value is mandatory. The other options are as described in Section 12.4.1, “Starting the X.400 Channel from the command line to pull messages”.

12.4.3 Starting the X.400 channel as a static responder

The X.400 channel can be run as a static responder, which may be useful for test purposes. In this mode, the x400p1 channel program takes note of the `ininfo` field of the channel

configuration. This field can be found in the **x400p1 Channel Properties/Program** window. If this is set to the value `sloppy`, no checking of MTA name and password is done for any inbound connection on that channel.

The following options can be included in the startup command:

```
x400p1 [-R] [-te|d] [-r|-s] [-f<logname>] [-o|-n] [-c<channel>]
      [-l<addr>]
```

`-R`
: Start as a responder (the default).

`-o`
: Use the old 1984 compatible stack.

`-c <channel>`
: Claim to be the given channel name on input. You may have a number of X.400 channels, selected initially by T-Selector or network address. If you wish to split up traffic in this way, possibly for authorization reasons, you should set the channel name in the *isoservices* file. By default the channel is assumed to be the name of the program itself.

  A better way to split traffic would be to use channel pairing, i.e. separate channels could be configured to handle inbound and outbound traffic. The channel key field could be used to identify the inbound or outbound channel. For example, the inbound channel for communicating with a specific MTA (site3) could be configured with the following values:

```
name=X.400site3
     program=x400p1x400p1
     key=X.400in88
```

  while the outbound channel could be configured with the values:

```
name=X.400site3
     program=x400p1
     key=X.400out88
```

`-l <addr>`
: Use the presentation address, *<addr>*, to listen for incoming connections, and spawn a child process to handle the connection. Note, this option can only be used when the channel is started from the command line, i.e. not being run from the tsapd or iaed.

The other options are as described in Section 12.4.1, “Starting the X.400 Channel from the command line to pull messages”.

Chapter 13 Audit Database

This section covers in detail the Audit Database, in particular the Audit Records and Keys that appear in the Audit Logs, and whether they are used in the Audit Database Schema.

13.1 Audit Database Records

13.1.1 Generic Message Logging

13.1.1.1 Audit DB Record: Msgin

Msgin records hold per-message information on submission or transfer-in.

Table 13.1. Audit DB Record: Msgin

Key	Description	DB
chan	The channel associated with the action.	Y
content-type	The message content type.	Y
deferred-time	The deferred delivery time set in the message.	Y
disp-notif-to	The value of the Disposition-Notification-To field in an Internet message.	Y
dtg	Date Time Group string in ACP 121 format.	Y
envid	The SMTP ENVID extension associated with the message.	Y
expires	MMHS Expiry Date Indication.	Y
mmhs-type	MMHS Content Type.	Y
msgid	Internet message ID.	Y
mta	Associated MTA information.	Y
nrecip	Number of envelope recipient.	Y
nreprecip	Number of reported recipient.	Y
originator-error		N
orig-sender	Original sender address.	Y
p1msgid	X.400 MTS Identifier.	Y
priority	Message priority (as internal value).	Y
qid	ID for message in queue.	Y
queued-time	Time the message was entered in the local queue.	Y
sender	Sender (originator) address.	Y
sics	List of Subject Information Codes.	Y
size	Size of message content.	Y
subject	Subject string	Y
subjectmsgid	MTS identifier of subject message.	Y
subjecttrace	Subject-trace-information from report content.	Y
submit-time	Date/time message was submitted.	Y
subtype	Message subtype: report type for multipart/report.	Y
trace	Trace information from envelope.	Y

Key	Description	DB
type	Message Type.	Y
unid	Unique Identifier.	Y

13.1.1.2 Audit DB Record: ok, resubmit, redirect

- ok: Indicates acceptance of a recipient on submission or transfer in.
- Resubmit: Results of recalculating the routing for a recipient.
- Redirect: An operator initiated redirection for a recipient.

Table 13.2. Audit DB Record: ok, resubmit, redirect

Key	Description	DB
auth	A string giving information from the authorization process.	Y
chan	The channel associated with the action.	Y
dn	The recipient's Directory Name.	Y
hold		N
in-recip	The recipient address as received in protocol.	Y
mreq	MTA report requests.	Y
mta	Associated MTA information.	Y
nreq	Notification requests.	Y
orig-recip	Original recipient address.	Y
qid	ID for message in queue.	Y
recip	Recipient address.	Y
rno	Internal recipient number.	Y
unid	Unique Identifier.	Y
ureq	User specified report requests.	Y
xno	Envelope recipient number.	Y

13.1.1.3 Audit DB Record: rrecip-pos, rrecip-neg

These records contain delivery or non-delivery information for a reported-recipient from an X.400 Report.

Table 13.3. Audit DB Record: rrecip-pos, rrecip-neg

Key	Description	DB
arrival-time	The time of subject message arrival.	Y
del-time (pos only)	The delivery time reported in an X.400 Report for the subject message.	Y
del-type (pos only)	The type of user for an X.400 Reported recipient	Y
diag (neg only)	A string reporting the diagnostic for non-delivery.	Y
origintrecip	Originally intended recipient address.	Y
qid	ID for message in queue.	Y
reason (neg only)	Non delivery reason.	Y
recip	Recipient address.	Y
suppinfo	Supplementary information for acknowledgements.	Y
unid	Unique Identifier.	Y
xno	Envelope recipient number.	Y

13.1.1.4 Audit DB Record: ACDFfail

Records a failure when checking a label against a clearance. Occurs when clearance checks are used in authorization.

Table 13.4. Audit DB Record: ACDFfail

Key	Description	DB
chan	The channel associated with the action.	Y
error		Y
mta	Associated MTA information.	Y
qid	ID for message in queue.	N
type	Message Type.	Y
unid	Unique Identifier.	Y

13.1.1.5 Audit DB Record: Error

An error condition resulting in non-delivery for the recipient.

Table 13.5. Audit DB Record: Error

Key	Description	DB
chan	The channel associated with the action.	Y
diag	A string reporting the diagnostic for non-delivery.	Y
info	Additional information.	Y
mta	Associated MTA information.	Y
qid	ID for message in queue.	N
reason		Y
recip	Recipient address.	Y
rno	Internal recipient number.	Y
status	Status of operation.	Y
unid	Unique Identifier.	Y

13.1.1.6 Audit DB Record: Archive

Records the archive file used to archive a message.

Table 13.6. Audit DB Record: Archive

Key	Description	DB
file	Name of the archive file.	Y
index		Y
qid	ID for message in queue.	N
tid	Transaction ID.	Y
unid	Unique Identifier.	Y

13.1.1.7 Audit DB Record: Label, Outlabel

Label and Outlabel records hold onformation about Security Label on oncoming messages (Label) and Outgoing messages (Outlabel). NB Outlabel records are only generated if specifically enabled.

Table 13.7. Audit DB Record: Label, Outlabel

Key	Description	DB
chan	The channel associated with the action.	Y
classif	Security Label CLssification.	Y
error		Y
qid	ID for message in queue.	N
textlabel	String of ACP127 form of Classification.	N
tid (Outlabel only)	Transaction ID.	N
unid	Unique Identifier.	Y
where	Where the security label was located.	Y
xmllabel	Value of security label expressed in XML.	Y

13.1.1.8 Audit DB Record: SignVerifyOK, SignVerifyNOTOK, SignVerifyUnknown, SignVerifyWarn

Sign records report information of message signatures.

- SignVerifyNOTOK: A failure when checking a message signature. Occurs when signature checking is enabled in authorization.
- SignVerifyOK: A success when checking a message signature.
- SignVerifyUnknown: TBS.
- SignVerifyWarn: A warning condition when checking a message signature.

Table 13.8. Audit DB Record: Sign

Key	Description	DB
==alt-name==		Y
chan	The channel associated with the action.	Y
issuer	The issuer DN from the signing certificate.	Y
qid	ID for message in queue.	N
serial	Signing Certificate’s serial number.	Y
signing-time	Time message was signed.	Y
status	Status of operation.	Y
subject	Subject string	Y
unid	Unique Identifier.	Y

13.1.1.9 Audit DB Record: DSN-msg

Per-message information from a DSN.

Table 13.9. Audit DB Record: DSN-msg

Key	Description	DB
arrival-time	The time of subject message arrival in a report or DSN.	Y
envid	The SMTP ENVID extension associated with the message.	Y
from-mta	The MTA from which the subject message was received as reported in a DSN.	Y
gateway	The gateway MTA generating a DSN.	Y
msgid	Internet message ID.	Y

Key	Description	DB
mta	Associated MTA information.	Y
nreprecip	Number of reported recipient.	Y
qid	ID for message in queue.	Y
unid	Unique Identifier.	Y

13.1.1.10 Audit DB Record: DSN-recip

Per-recipient information from a DSN.

Table 13.10. Audit DB Record: DSN-recip

Key	Description	DB
action	The event which led to the DSN being created. One of "failed" / "delayed" / "delivered" / "relayed" / "expanded.	Y
attempt	The time of the last attempt (from a Warning DSN only).	Y
diag	A string reporting the diagnostic for non-delivery.	Y
logid	Value of final-log-id.	Y
mta	Associated MTA information.	Y
orig-recip	Original recipient address.	Y
qid	ID for message in queue.	N
recip	Recipient address.	Y
retry	Time at which attempts will stop.	Y
status	Status of operation.	Y
unid	Unique Identifier.	Y

13.1.1.11 Audit DB Record: MDN

Disposition information from a MDN.

Table 13.11. Audit DB Record: MDN

Key	Description	DB
action		Y
gateway	The gateway MTA generating an MDN.	Y
msgid	Internet message ID.	Y
orig-recip	Original recipient address.	Y
qid	ID for message in queue.	N
recip	Recipient address.	Y
status	Status of operation.	Y
suppinfo	Supplementary information for acknowledgements.	Y
ua	User Agent identification.	Y
unid	Unique Identifier.	Y

13.1.1.12 Audit DB Record: IPM

Information from within an X.400 IPM.

Table 13.12. Audit DB Record: IPM

Key	Description	DB
dtg	Date Time Group string in ACP 121 format.	Y

Key	Description	DB
		Y
expires	MMHS Expiry Date Indication.	Y
ipmid-dn	The DN from the IPM identifier.	Y
ipmid-ora	The OR-address from the IPM identifier.	Y
ipmid-str	The string from the IPM identifier.	Y
mmhs-type	MMHS Content Type.	Y
qid	ID for message in queue.	N
subject	Subject string	Y
unid	Unique Identifier.	Y

13.1.1.13 Audit DB Record: IPN

Receipt or non-receipt information from an X.400 IPN.

Table 13.13. Audit DB Record: IPN

Key	Description	DB
discard	The discard reason in an X.400 IPN.	Y
intend-dn	The DN for the intended recipient field .	Y
intend-ora	The OR-address for the intended recipient.	Y
ipmid-dn	The DN from the subject IPM identifier.	Y
ipmid-ora	The OR-address from the subject IPM identifier .	Y
ipmid-str	The string from the subject IPM identifier.	Y
mode	Receipt mode.	Y
orig-dn	Originator DN.	Y
orig-ora	Originator OR Address	Y
qid	ID for message in queue.	N
reason	Non-delivery reason.	Y
receipt-time	Time message was received.	Y
returned-ipm	IPN contains returned IPM.	Y
suppinfo	Supplementary information for acknowledgements.	Y
unid	Unique Identifier.	Y

13.1.1.14 Audit DB Record: Trans, Deliv, Done, Quarantine, Discard

Message procecssing.

- Trans: Message transferred for recipient to another MTA.
- Deliv: Message delivered for recipient.
- Done: Recipient finished, following report/DSN generation.
- Quarantine: Message for recipient quarantined, specifying the quarantine file.
- Discard: Message discarded for recipient. No non-delivery generated.

Table 13.14. Audit DB Record: Trans, Deliv, Done, Quarantine, Discard

Key	Description	DB
action-time	The time of a message transfer.	Y
chan	The channel associated with the action.	Y
delivery-time	The time the message was delivered.	Y

Key	Description	DB
discard-reason (Discard only)	The local reason for discarding a message.	Y
qfile (Quarantine only)	Quarantine filename.	Y
qid	ID for message in queue.	N
qtime	Date/time message was in queue for recipient.	Y
recip	Recipient address.	Y
in-recip	The recipient address as received in protocol.	Y
orig-recip	Original recipient address.	Y
report	Indicates positive report or DSN to be generated.	Y
rno	Internal recipient number.	Y
tid	Transaction ID.	Y
unid	Unique Identifier.	Y

13.1.1.15 Audit DB Record: Msgout

Message transfer-out or delivery.

Table 13.15. Audit DB Record: Msgout

Key	Description	DB
acp127tid		Y
chan	The channel associated with the action.	Y
mta	Associated MTA information.	Y
nrecip	Number of envelope recipient.	Y
qid	ID for message in queue.	N
sender	Sender (originator) address.	Y
size	Size of message content.	Y
subject	Subject string	Y
tid	Transaction ID.	Y
ttime	Time taken for transfer.	Y
unid	Unique Identifier.	Y

13.1.2 Message Release

13.1.2.1 Audit DB Record: Release

Records the release of a message from quarantine.

This record is not recorded in the audit database.

Table 13.16. Audit DB Record: Release

Key	Description	DB
qid	ID for message in queue.	
recip	Recipient address.	
rno	Internal recipient number.	
unid	Unique Identifier.	

13.1.3 Badmsg from ACP127, ACP142, P1

13.1.3.1 Audit DB Record: Badmsg

Indicates that data was received by the inbound channel which could not be interpreted as a message.

This record is not recorded in the audit database.

Table 13.17. Audit DB Record: Badmsg

Key	Description	DB
acp127circ	ACP127 circuit name	
chan	The channel associated with the action.	
file	Name of the dead letter file.	
mta	Associated MTA information.	
reason	String giving the reason the message was regarded as bad.	
type	Message Type.	

13.1.4 P1 Connections

13.1.4.1 Audit DB Record: P1InitConnOK, P1InitConnFail, P1RespConnOK, P1RespConnFail

P1 connections records, initiator and responder, inbound and outbound, successes and failures.

Table 13.18. Audit DB Record: P1InitConnOK, P1InitConnFail, P1RespConnOK, P1RespConnFail

Key	Description	DB
actid	Activity ID of previous association.	Y
actno	Activity number.	Y
appcon	Application context (0 = 84; 1 = X.410; 2 = Normal).	Y
auth_req	Their authentication requirements.	Y
bindtype	strong or simple.	Y
chan	The channel in use.	Y
ckpoint	RTSE checkpoint size.	Y
dialogmode	TWA or monologue.	Y
fail_reason	Reason for a P1 connection failure.	Y
our_auth_req	Our authentication requirements.	Y
ourissuer	The issuer DN of the local signing certificate.	Y
ourmtaname	The local MTA's MTAname used in the bind.	Y
ourpa	The local channel's presentation address.	Y
oursangdi	Local global domain identifier.	Y
oursanmta	Local MTAname.	Y
ourserial_num	Issuer of local signing certificate.	Y
oursubject	Subject of local signing certificate.	Y
recov	Association being recovered.	Y
res_msgkey	Key for message on recovery.	Y
res_p1len	Length of message being transferred on recovery.	Y

Key	Description	DB
res_priority	Message priority on recovery.	Y
rtse_type	The RTSE type for the connection.	Y
rts_flags	Internal RTS flags.	Y
rtsid	RTS identifier.	Y
theiraet	The peer’s DN from the AET.	Y
their_calling_addr	The peer’s address used for the connection.	Y
theirissuer	The issuer DN of peer's signing certificate.	Y
theirmtaname	The peer’s MTAname.	Y
theirpa	The peer’s presentation address.	Y
theirsangdi	Peer’s global domain identifier.	Y
theirsanmta	Peer’s MTAname.	Y
theirserial_num	Serial number of peer’s signing certificate.	Y
theirsubject	Subject of peer’s signing certificate.	Y
tokengdi	GDI in token.	Y
tokenmta	MTAname in token.	Y
window	RTSE window size.	Y

13.1.4.2 Audit DB Record: P1InitDiscOK, P1InitAbort, P1InitReject, P1Unknown, P1RespDiscOK, P1RespAbort

P1 disconnection records, initiator and responder, inbound and outbound, successes and failures.

Table 13.19. Audit DB Record: P1InitDiscOK, P1InitAbort, P1InitReject, P1Unknown, P1RespDiscOK, P1RespAbort

Key	Description	DB
bindtype	strong or simple.	Y
chan	The channel in use.	Y
datatrans	Data transfer amount.	Y
discstatus	Disconnect status.	Y
fail_reason	Reason for a P1 connection failure.	Y
ourissuer	The issuer DN of the local signing certificate.	Y
ourmtaname	The local MTA’s MTAname used in the bind.	Y
ourpa	The local channel’s presentation address.	Y
oursangdi	Local global domain identifier.	Y
oursanmta	Local MTAname.	Y
ourserial_num	Issuer of local signing certificate.	Y
oursubject	Subject of local signing certificate.	Y
rta_reason	Disconnect reason from RTSE.	Y
sa_fail_reason	Strong authentication failure reason.	Y
sn_ack	RTS ack number.	Y
sn_sent	RTS session serial number.	Y
theirissuer	The issuer DN of peer's signing certificate.	Y
theirmtaname	The peer’s MTAname.	Y
theirpa	The peer’s presentation address.	Y

Key	Description	DB
theirsangdi	Peer's global domain identifier.	Y
theirsanmta	Peer's MTAname.	Y
theirserial_num	Serial number of peer's signing certificate.	Y
theirsubject	Subject of peer's signing certificate.	Y
tokengdi	GDI in token.	Y
tokenmta	MTAname in token.	Y

13.1.5 P3 Connections

13.1.5.1 Audit DB Record: P3InitConnOK, P3InitConnFail, P3RespConnOK, P3RespConnFail

P3 connection records, initiator and responder, inbound and outbound, successes and failures.

Table 13.20. Audit DB Record: P3InitConnOK, P3InitConnFail, P3RespConnOK, P3RespConnFail

Key	Description	DB
addr	Calling O/R address.	Y
admin		N
chan	The channel in use.	Y
dn	Calling Directory Name.	Y
fail_reason	Reason for a P3 connection failure.	Y
twa	Bi-directional connection.	Y
type	Message Type.	Y

13.1.5.2 Audit DB Record: P3InitDiscOK, P3InitAbort, P3RespDiskOK, P3RespAbort

P3 disconnection records, initiator and responder, inbound and outbound, successes and failures.

Table 13.21. Audit DB Record: P3Other

Key	Description	DB
addr	Calling O/R address.	Y
chan	The channel in use.	Y
discstatus	Disconnect status.	Y
dn	Calling Directory Name.	Y
type	Message Type.	Y

13.1.5.3 Audit DB Record: P3ConnectFrom

Reports when an inbound P3 protocol connection is received.

This record is not recorded in the audit database.

Table 13.22. Audit DB Record: P3ConnectFrom

Key	Description	DB
chan	Channel name.	
calling_addr	The calling Presentation Address.	

Key	Description	DB
ourpa	The Presentation Address of the channel.	

13.1.6 ACP127

13.1.6.1 Audit DB Record: ACP127addrMapFail

This record is not recorded in the audit database

Table 13.23. Audit DB Record: ACP127addrMapFail

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127id	Station serial Number	
acp127tsn	ACP127 Transmission Sequence Number	
chan	The channel associated with the action.	
origPla	Originating PLA	
origRi	Originating Routing Indicator	
replacement	Replacement Address	

13.1.6.2 Audit DB Record: ACP127Fill

This record is not recorded in the audit database

Table 13.24. Audit DB Record: ACP127Fill

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
action	Action string (e.g. fillsend)	
chan	The channel associated with the action.	
station	Station RI	

13.1.6.3 Audit DB Record: ACP127Recap

This record is not recorded in the audit database

Table 13.25. Audit DB Record: ACP127Recap

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
action	action string (e.g. recapsend)	
chan	The channel associated with the action.	
count		
from	Date Time from	
mta	Associated MTA information.	
rno	Internal recipient number.	
station	RI of the station	
to	Date Time to	

13.1.6.4 Audit DB Record: ACP127Silence

This record is not recorded in the audit database

Table 13.26. Audit DB Record: ACP127Silence

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
chan	The channel associated with the action.	
station	Station RI	

13.1.6.5 Audit DB Record: ACP127out

Table 13.27. Audit DB Record: ACP127out

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	Y
acp127circ	ACP127 circuit name	N
acp127id	Station serial Number	Y
acp127rerunid	Retransmission ID of the message	Y
acp127segno	This is the segment id to identify a message segment when a long message is segmented	N
acp127tid	This is used to correlate acp127out records with msgout records	Y
acp127totalseg	The total number of message segments of original message	N
acp127tsn	ACP127 Transmission Sequence Number	Y
action	The addresses the message is actioned to	Y
chan	The channel associated with the message.	Y
dtg	Date Time Group string in ACP 121 format.	Y
info	The addresses the message is sent as an information	Y
mmhs-type	MMHS Content Type.	Y
mta	Associated MTA information.	Y
opparam	Paramaters associated with operating signal	Y
opsig	Operating signal code	Y
orig-pla	Originating PLA	Y
service	Determines if the record belongs to a service message	Y
subject	Subject string	Y
unid	Unique Identifier.	Y

13.1.6.6 Audit DB Record: ACP127Rejected

This record is not recorded in the audit database

Table 13.28. Audit DB Record: ACP127Rejected

Key	Description	DB
acp127id	Station serial Number	
action	The addresses the message is actioned to	
chan	The channel associated with the message.	
dtg	Date Time Group string in ACP 121 format.	
info	The addresses the message is sent as an information	

Key	Description	DB
mmhs-type	MMHS Content Type.	
opsig	Operating Signal Code	
orig-pla	Originating PLA	
qid	ID for message in queue.	
reason	Non-delivery reason, or reason message bad.	
service	Identify a service message	
subject	Subject string	
unid	Unique Identifier of message in the MTA	
zpw	Expiry time	

13.1.6.7 Audit DB Record: ACP127in

Table 13.29. Audit DB Record: ACP127in

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	Y
acp127circ	ACP127 circuit name	N
acp127id	Station serial Number	Y
acp127rerunid	Retransmission ID of the message	Y
acp127segno	This is the segment id to identify a message segment when a long message is segmented	Y
acp127totalseg	The total number of message segments of original message	Y
acp127tsn	ACP127 Transmission Sequence Number	Y
action	The addresses the message is actioned to	Y
chan	The channel associated with the action.	Y
dtg	Date Time Group string in ACP 121 format.	Y
info	The addresses the message is sent as an information	Y
mmhs-type	MMHS Content Type.	Y
mta	Associated MTA information.	Y
opparam	Paramaters associated with operating signal	Y
opsig	Operating signal code	Y
orig-pla	Originating PLA	Y
qid	ID for message in queue.	N
service	Determines if the record belongs to a service message	Y
subject	Subject string	Y
submissionMsg	String associated with message submission(e.g. discarded duplicate)	Y
submissionType	String associated with message submission type(e.g. duplicate)	Y
submissionUser	User id who submitted the message	N
unid	Unique Identifier.	Y

13.1.6.8 Audit DB Record: ACP127QueueDelete

This record is not recorded in the audit database

Table 13.30. Audit DB Record: ACP127QueueDelete

Key	Description	DB
acp127circ	ACP127 circuit name	
action	The addresses the message is actioned to	
chan	The channel associated with the action.	
entry	Identifier number in the queue on which operation was performed	
mta	Associated MTA information.	
queue	The name of the queue	
user	User if of the person performing the action	

13.1.6.9 Audit DB Record: ACP127QueueAddition

This record is not recorded in the audit database

Table 13.31. Audit DB Record: ACP127QueueAddition

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
acp127id	Station serial Number	
acp127tsn	ACP127 Transmission Sequence Number	
action	The addresses the message is actioned to	
chan	The channel associated with the action.	
dtg	Date Time Group string in ACP 121 format.	
entry	Identifier number in the queue on which operation was performed	
error	Error string	
info	Information string	
mmhs-type	MMHS Content Type.	
mta	Associated MTA information.	
opsig	Operating signal code	
orig-pla	Originating PLA	
queue	Name of the queue	
service	Determines if the record belongs to a service message	
size	Size of message content.	
subject	Subject string	

13.1.6.10 Audit DB Record: ACP127AuditEvent

This record is not recorded in the audit database

Table 13.32. Audit DB Record: ACP127AuditEvent

Key	Description	DB
acp127circ	ACP127 circuit name	
action	The addresses the message is actioned to	
chan	The channel associated with the action.	
mta	Associated MTA information.	
unid	Unique Identifier.	
user	Identifier of the user associated with the operation	

13.1.6.11 Audit DB Record: ACP127Conn

This record is not recorded in the audit database

Table 13.33. Audit DB Record: ACP127Conn

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
addr		
chan	The channel associated with the action.	
direction	connection direction(in/out)	
error	Error string	
mta	Associated MTA information.	
reason	Reason string	
station	Station RI	
status	Status of operation (e.g.connectFailed)	

13.1.6.12 Audit DB Record: ACP127FlashAck

This record is not recorded in the audit database

Table 13.34. Audit DB Record: ACP127FlashAck

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
chan	The channel associated with the action.	
direction		
expected		
got		
mta	Associated MTA information.	
station		
status	Status of operation.	

13.1.6.13 Audit DB Record: ACP127SM

This record is not recorded in the audit database

Table 13.35. Audit DB Record: ACP127SM

Key	Description	DB
acp127circ	ACP127 circuit name	
chan	The channel associated with the action.	
mta	Associated MTA information.	
params	parameters	
station	Station RI	
type	Message Type.	
user	User ID	

13.1.6.14 Audit DB Record: ACP127Config

This record is not recorded in the audit database

Table 13.36. Audit DB Record: ACP127Config

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
chan	The channel associated with the action.	
change	String describing the modification	
mta	Associated MTA information.	
station	Station RI	
user	User ID	

13.1.6.15 Audit DB Record: ACP127ManualOp

This record is not recorded in the audit database

Table 13.37. Audit DB Record: ACP127ManualOp

Key	Description	DB
acp127chan	This is the channel designator associated with the peer	
acp127circ	ACP127 circuit name	
chan	Associated channel	
mta	Associated MTA information.	
operation	String describing the operation	
station	Station RI	
unid	Unique Identifier.	
user	User Identifier performing the operation	

13.1.6.16 Audit DB Record: ACP127Monitor

This record is not recorded in the audit database

Table 13.38. Audit DB Record: ACP127Monitor

Key	Description	DB
authorisation	String describing the authorisation (e.g. full)	
chan	associated channel	
monitor		
peeraddr	address of peer	
type	Message Type.	
username	User Identifier	

13.1.7 OTAM

13.1.7.1 Audit DB Record: OTAMConn

This record is not recorded in the audit database

Table 13.39. Audit DB Record: OTAMConn

Key	Description	DB
acp127circ	ACP127 circuit name	
chan		
config		
device		
driver		
mta	Associated MTA information.	
reason		
status	Status of operation.	

13.1.8 SMTP

13.1.8.1 Audit DB Record: ConnectFrom

Incoming SMTP connection.

Table 13.40. Audit DB Record: ConnectFrom

Key	Description	DB
chan	Channel being used.	Y
host	hostname of peer.	Y
ip	IP address of peer.	Y

13.1.8.2 Audit DB Record: Disconnect, Reject, BadSyntax

SMTP connection errors.

Table 13.41. Audit DB Record: Disconnect, Reject, BadSyntax

Key	Description	DB
chan	Channel being used.	Y
helo	Value sent in HELO/EHLO.	Y
host	hostname of peer.	Y
ip	IP address of peer.	Y
reason	reason for failure.	Y

13.1.8.3 Audit DB Record: BadSender, BadRecip, BadData

SMTP protocol errors.

- BadSender: Invalid sender address in MAIL command.
- BadRecip: Invalid recipient address is RCPT command.
- BadData: Invalid message content.
- BadVerify: Invalid address in VRFY command.

Table 13.42. Audit DB Record: BadSender, BadRecip, BadData

Key	Description	DB
addr	Address received.	Y
chan	Channel being used.	Y
helo	Value sent in HELO/EHLO.	Y

Key	Description	DB
host	hostname of peer.	Y
ip	IP address of peer.	Y
reason	Reason for failure.	Y
sender	MAIL command address.	Y

13.1.8.4 Audit DB Record: AuthFail

SMTP Authentication errors SMTP AUTH failed.

Table 13.43. Audit DB Record: AuthFail

Key	Description	DB
chan	Channel being used.	Y
helo	Value sent in HELO/EHLO.	Y
host	hostname of peer.	Y
ip	IP address of peer.	Y
reason	Reason for failure	Y
user	Userid in AUTH command.	Y

13.1.8.5 Audit DB Record: AuthOK

SMTP Authentication: SMTP AUTH succeeded.

Table 13.44. Audit DB Record: AuthOK

Key	Description	DB
addr	Address received.	Y
chan	Channel being used.	Y
helo	Value sent in HELO/EHLO.	Y
host	hostname of peer.	Y
ip	IP address of peer.	Y
mech	Mechanism used for SMTP AUTH.	Y
user	Userid in AUTH command.	Y

13.1.8.6 Audit DB Record: StartTLSOK, StartTLSFail

SMTP TLS Authentication: STARTTLS failed/succeeded.

Table 13.45. Audit DB Record: StartTLSFail

Key	Description	DB
chan	Channel being used.	Y
fips	Set to “yes” if FIPS140 mode enabled.	Y
helo	Value sent in HELO/EHLO.	Y
host	hostname of peer.	Y
ip	IP address of peer.	Y
peer	Name of TLS peer.	Y

13.1.8.7 Audit DB Record: Unknown

This record is not recorded in the audit database

P1Unknown: Unknown connection termination.

Table 13.46. Audit DB Record: Unknown

Key	Description	DB
chan		
helo	Value sent in HELO/EHLO.	
host		
ip	IP address of peer.	

13.1.9 ACP142

13.1.9.1 Audit DB Record: ACP142out

A message has been transferred out over ACP142.

Table 13.47. Audit DB Record: ACP142out

Key	Description	DB
chan		Y
co		Y
msid		Y
ndest		Y
percent		Y
qid	ID for message in queue.	Y
unid	Unique Identifier.	Y

13.1.9.2 Audit DB Record: ACP142in

A message has been transferred in over ACP142.

Table 13.48. Audit DB Record: ACP142in

Key	Description	DB
chan		Y
co		Y
msid		Y
qid		Y
source		Y
unid	Unique Identifier.	Y

13.1.10 Checking and CCCP

13.1.10.1 Audit DB Record: Check

Records the results of checking a message. There is a record for each recipient.

Table 13.49. Audit DB Record: Check

Key	Description	DB
action		Y
chan		Y
info	String containg additional information.	Y
notify	Indicates change in NOTIFY parameter in checking.	Y

Key	Description	DB
qid		N
rno	Internal recipient number.	Y
rule	Checking rule which applied.	Y
score	Checking score.	Y
unid	Unique Identifier.	Y

13.1.11 Generic Service start/stop

13.1.11.1 Audit DB Record: Service

Reports a service status change.

This record is not recorded in the audit database.

Table 13.50. Audit DB Record: Service

Key	Description	DB
name		
state		
version		

13.1.12 Qmgr SOM

13.1.12.1 Audit DB Record: Login, Logfail, Logout

Reports login attempts on a SOM server.

This record is not recorded in the audit database.

Table 13.51. Audit DB Record: Login, Logfail, Logout

Key	Description	DB
addr	Address received.	
authzid		
connid		
context		
mech	Mechanism used for SMTP AUTH.	
reason	Reason for failure	
user	Userid in AUTH command.	

13.1.12.2 Audit DB Record: UserEvent

Reports SOM server events from a logged in client.

This record is not recorded in the audit database.

Table 13.52. Audit DB Record: UserEvent

Key	Description	DB
connid		
context		
info		
type	Message Type.	

Key	Description	DB
user	Userid in AUTH command.	

13.1.13 Profiler

13.1.13.1 Audit DB Record: ProfilerMatch

Reports profiler rules that a message matched with and the action, info and recipient addresses for the message.

This record is not recorded in the audit database.

Table 13.53. Audit DB Record: ProfilerMatch

Key	Description	DB
unid	Unique Identifier.	
qid	ID for message in queue.	
chan	Channel being used.	
tid	Transaction ID.	
rule	Profiler rules the message matched with.	
recip	Recipient addresses.	
action	Action addresses.	
info	Info addresses.	

13.1.13.2 Audit DB Record: ManualAction

Reports when an action was taken by the operator of the Manual Profiler and includes a string describing the action.

This record is not recorded in the audit database.

Table 13.54. Audit DB Record: ManualAction

Key	Description	DB
unid	Unique Identifier.	
qid	ID for message in queue.	
chan	Channel being used.	
tid	Transaction ID.	
operator	Login name of operator who took the action.	
action	String describing action taken by operator.	