`prefix=number:` the number to prepended to the remote fax machine number. This is useful for a fax machine connected to an internal network where a prefix is required to get external numbers. If `prefix` is not present no number is prepended.

`nattempts=number:` the number of times the channel will attempt to connect to the remote fax machine in one invocation. The default is three times.

`sleep=numSecs:` the number of seconds to sleep between connection attempts. The default is 30 seconds.

`softcar=used:` indicates that the software carrier detect package is in place.

The outbound channel's behaviour is also altered by the fashion in which the recipient was addressed. If the recipient was addressed directly via a telephone number, e.g `FAX=999`, and this value was not found in the fax table, then the fax machine will generate a negative delivery report if it cannot delivery the message in the first invocation (which consists of `nattempts` connection attempts) providing that a connection with the remote site is made. If the value is found in the fax table, then the fax channel will not bounce the message if the first invocation fails to transmit it. Instead it will revert to the usual PP timeout method for nondelivery.

The outbound channel contains special processing for the telephone numbers of the remote machine. This is as follows:

- If the number starts with the character `+`, the international prefix is prepended to the number. The international prefix is stored in the fax table.
- Any "`P`" characters in the number cause the fax transmission channel to pause at those positions while dialing.
- If the number is zero, the fax channel will print the fax on the local fax machine rather than send it to a remote fax.

The Inbound Reception Daemon

The Panasonic Systemfax's inbound daemon's behaviour can be altered by several key/value pairs in the `ininfo` tailor field:

`in=device:` the inbound device on which to listen for incoming fax. By default it will try to use `/dev/faxin`.

`master=address:` the address to send incoming faxes to. This will default to `faxmaster`.

`subject=string:` the subject to append to the messages encapsulating the incoming faxes. This defaults to the string "Inbound fax message".

`softcar=used:` indicates that the software carrier detect package is in place.

By default the inbound daemon will run as a channel named `fax`. This can be altered by the command line argument `-c` (e.g. `ps250d -c fax-in`)

6.2 ASN.1 Filter

The ASN.1 Filter *asn*, is used in conjunction with the general filter controller *fcontrol*, and is located in the *formdir* directory.

It is a general body part conversion filter and can do a variety of conversions depending on its specified options. These options are:

- Unpack ASN.1 encodings of bodyparts.
- Perform a character set conversion on the body part.
- Pack the body part into ASN.1

6.2.1 Runtime Options

The *asn* filter can take a number of arguments. All of these are optional. They are as follows:

`-ia encoding:` The type of ASN.1 decoding to be done on input. If the encoding is specified as "none" no decoding will be done (the default).

`-oa encoding:` The type of ASN.1 encoding to be done on ouput. If "none" is specified no encoding will be done on output (the default).

`-is charset:` The input character set.

`-os charset:` The character set to be produced. If this is the same as the `-is` parameter, no conversion will be performed.

`-x:` Use X.408 encodings (the default).

`-m:` Use Mnemonic encodings.

`-l:` Add a trailing newline to each ASN.1 block decoded. This should be used with care.

The character sets supported at present are:

none: Refers to no character set.

ia5: This is unencoded ascii text.

generaltext: The ISO general test format.

motis-86-6937: The MOTIS defined 6937 character set.

teletex: The T.61 character set.

```
chan iso6937toia5
        prog=fcontrol,
        show="iso6937 -> ia5 conversion filter",
        bptin=iso6937, bptout=ia5,
        type=shaper, conv=loss, cost=99,
        outinfo="asn -is iso_6937-2-add -os ccitt_x.408_ia5irv"

chan ia5toiso6937
        prog=fcontrol,
        show="ia5 -> iso6937 conversion filter",                    10
        bptin=ia5, bptout=iso6937,
        type=shaper, conv=loss, cost=99,
        outinfo="asn -is ccitt_x.408_ia5irv -os iso_6937-2-add
                -ia ia5 -oa motis-86-6937"

chan iso6937toteletex
        prog=fcontrol,
        show="iso6937 -> teletex conversion filter",
        bptin=iso6937, bptout=ttx,
        type=shaper, conv=loss, cost=99,                            20
        outinfo="asn -is iso_6937-2-add -os t.61-8bit
                 -ia motis-86-6937 -oa teletex"
```

Figure 6.3: Examples of the asn filter

6.2.2 Tailoring

The following fields need to be set within the tailor file.

`prog:` This should be set to the value *fcontrol*, as this program is always run under *fcontrol*.

`bptin:` This should be set to the inbound character set.

`bptout:` This should be set to specified character set to be produced.

`type:` set to `shaper`.

`conv:` set to either `conv` if the conversion is reversible or `loss` if the conversion looses information.

`cost:` set this to a highish value (say 99) if `conv` is `loss`.

`outinfo:` set to the ASN.1 filter name *asn* followed by its required options.

Tailoring Examples

Figure 6.3 shows examples of tailoring the *asn* filter. The first converts iso6937 format into ascii, this conversion is not generally reversible, so `loss` is specified. The second example shows conversion of ia5 into iso6937. The third specifies a conversion from iso6937 into teletex.

Chapter 7

Managing PP

This chapter covers the tools which allow the system administrator to keep up to date with the status of the system and to take action to fix or avoid problems. It also gives some guidance on what is entailed in daily maintenance of the system, and how new users and MTAs may be added.

7.1 QMGR

The QMGR operates without ever reading the queue. The key ASN.1 structure is MsgStruct. This is the ASN.1 representation of the message as held by the queue manager. This is held in a sparse matrix, initially sorted by channel, host/user, and priority. The queue manager is not, inherently, a trusted process, as it does not access the queue directly. The protocol is listed in Appendix F in Figure F.1.

7.1.1 Message Submission

When *submit* is started, it asynchronously establishes an association with the QMGR. When a message is submitted to the queue, a ROS message is sent to the queue manager which will hold all of the information needed by the QMGR. This allows the message to be processed without the QMGR actually reading the queue.

7.1.2 Queue Reading

A ROS operation allows the queue to be read by calling up the *qmgrload* channel. This is used at startup and for occasional refresh of the queue. Note that when the queue is reread only new messages are added, messages that are already known to the QMGR are ignored.

7.1.3 Active Channel

An active channel is invoked by the QMGR, which starts a ROS association. The channel is synchronous. The first operation is ChannelInitialise, which identifies the channel to the process (which may act as several channels). Subsequent ChannelInitialise operations serve to change the role of

the channel. The ProcessMessage operation allows the QMGR to control the channel, which will access the queue directly.

7.1.4 Passive Channel

Passive channels have not yet been implemented.

7.1.5 Cleanups, Timeouts, Warnings and Debris

Message are deleted by the *msg-clean* channel. This is invoked after all recipients have been processed.

Messages are returned after a certain time by the QMGR invoking the *timout* channel. This process creates a delivery report and changes the status of the message to non-delivery.

Message that have been in the queue for a long period may generate warning messages to the originator. This is done via the *warning* channel.

Partially submitted messages and general debris are cleaned up quietly by debris channel after being in the queue for a certain amount of time. This channel is invoked at intervals.

7.1.6 QMGR Management and Tailoring

The QMGR is managed mostly through the MTA console interface. However, there are several options you can give when starting up the QMGR to change the mode of operation. These are listed here, in approximate order of usefulness:

`-m:` This switch sets the default maximum number of channels that can be run simultaneously. This parameter needs to be set carefully (see below).

`-c seconds:` This switch sets the default time to cache a channel that has failed. It defaults to 5 minutes. The cache value is multiplied by 1.5 to obtain a delay for failed MTAs, and multiplied by 2 for failed messages. Successive failures will multiply this value up to some maximum.

`-D:` This switch starts up the QMGR with all channels disabled. This is most useful to allow controlled startup of channels. Once started in this mode, channels can be gradually enabled by use of appropriate management tools.

`-d hours:` This switch sets the time between successive runs of the debris channel. This is the channel that removes extraneous bits of messages, etc. Its default value is about six hours.

`-l hours:` This switch sets the time between successive runs of the load channel. This channel is run at intervals to check consistency between the image of the queue in memory and that on disk.

`-t hours:` This switch sets the time between successive calls to the timeout channel. This channel is run at intervals to time out messages that have been in the queue too long.

`-C:` Do not change directory to the queue directory. Stay in the current directory.

`-b:` Do not run in the background, stay in the foreground.

`-T file:` Use this file as the pp *tailor* file.

`-X:` Turn up the debugging.

`-s:` Inhibit submission of messages. If this is set, the QMGR will ignore all incoming messages.

`-M:` This flag allows the *msg-clean* channel to run concurrently with the *qmgr-load* channel. There is a slight possibility of a race condition occuring if this is enabled, which may mean a message is not processed. The chance is slight however. If your queues are reasonably small (say ¡ 1000 messages normally) this flag should not be used. If you have large queues commonly, the load phase can take up to an hour, in which case allowing deletions during this time is useful.

The QMGR also responds to various signals (where available). A `SIGTERM` signal will cause the QMGR to gracefully shutdown. This is the same as QMGR control option. A `SIGUSR1` signal will increase the debugging level and a `SIGUSR2` will reset the level, or decrease it. All other signals default to their normal actions.

The loading on a given machine varies and it is controlled mainly through the maximum number of channels variable. Consequently, it is important to get this factor correct, otherwise response will suffer.

The main adjustable parameter is the maximum number of channels that can be run at any one time. This parameter should be set with care. It can be set by the above-mentioned flag at QMGR start time, or can be modified dynamically through the use of the MTA console. This parameter should be used in conjunction with the per channel maximum number of processes. The QMGR has a number of built in heuristics as to how useful running several invocations of a channel at once is. This can be overridden by the per-channel maximums, but these should be used sparingly. Suitable cases for imposing a per-channel maximum are the following:

- Channels which run very quickly. Such channels are usually reformatters that do their job quickly. Allowing only one or two instances of these channels can help loading in some cases. These channels can create a large load on the machine as they have no external influence and run as fast as the machine will let them.

- Special network channels. If you have a channel that uses a dedicated resource, then this may be a candidate for setting a maximum on.

Such a resource might be a modem or a non-multiplexed line. Clearly, if there is only one logical stream that a channel can use, starting multiple instances of such a channel is pointless.

Cases that should generally not have a per-channel maximum are the following.

- Network channels. Unless you have an amazingly fast network and instantaneous response from remote hosts, network channels do not usually create too much load. They tend to be self-throttling, as they spend much of their time waiting for I/O to complete or for response from the remote host.

- QMGR special channels. Such channels as QMGR-load and timeout are special to the QMGR and will only be invoked one at a time.

The maximum number of channels defaults to three, which is a reasonable number for a small machine that is doing other things besides mail. If you are running a dedicated mail server you will probably get better throughput with a larger number. However, it is worth spending some time watching what happens and increasing and decreasing this number via the MTA consoles to get a reasonable value. If you specify a value of 10, for instance, you may end up just wiping out any trace of performance on your machine, as the machine will just be thrashing between processes. On the other hand, 10 channels talking to 300 baud modems will probably not be noticeable. Once you have determined a reasonable value, add this to your pp.start script using the `-m` switch detailed above.

To begin with it is recommended that the default configuration with no per-channel maxima is used and that it is run for a while before adjustment is made.

7.2 MTA Console

The MTA console is a management tool that can be used to monitor and/or control a QMGR. This tool uses the Athena widget set on top of X11 release 4 (or greater).

7.2.1 The MTA Console's Display

The MTA console's display has two modes, the *monitor* mode and the *control* mode. Toggling between these modes is done via the appropriate button.

The top section of the display is common to both modes. This section contains a box within which the states of all the channels are displayed.

The *state* of an entity (channel, MTA or message) represents how "bad" that entity is, compared to the upperbound on "badness" (maximum badness). The badness of an entity can reflect various things like number of messages on that entity or the age of the oldest message on that entity. Section 7.2.4 describes what badness represents, how badness is calculated, and how this calculation can be tailored.

The Monitor Mode

The *monitor* mode provides a passive display of the overall state of the QMGR.

In this mode, the lower part of the display presents an ordered list of the states of the channels. The list is ordered on the basis of comparative badness. The list is laid out vertically with the worst state channel at the top.

The states of the "n worst state" MTAs on each channel are displayed to the right of the channel they are on. They are ordered on the basis of comparative badness with the worst state MTA being display directly to the right of the channel. How many of the MTAs on a channel are actually displayed depends on the heuristic being employed. The MTA console has a number of possible heuristics. These are described in Section 7.2.2.

The Control Mode

The *control* mode allows the user to focus on particular channels, MTAs or messages and control their passage through and their behaviour in the QMGR.

The amount of control the console can invoke depends on how the console connected to the QMGR. There are two types of connection the QMGR allows, unauthorised and authorised. With an authorised connection, the console can control the QMGR by disabling entities, enabling entities and so on. With an unauthorised connection, the console has no control over the QMGR. To get an authorised connection, the user has to specify a name and password when he or she requests the connection.

In the *control* mode, the lower part of the console's display presents information on the entity the user has focused on and allows the user to focus on other entities.

7.2.2 Configuring The MTA Console

This section details how the MTA console may be configured by users to suit their own individual tastes. There are three static mechanisms that can be used for this configuration:

1. command line arguments
2. user defaults
3. application defaults

As well as these static mechanisms, the MTA console contains a configuration menu. This menu can be used dynamically to configure many of the options described below.

Totals Configuration.

The MTA console displays the total number of messages and the total volume of messages in the queue. It also indicates how full the queue is by comparing these totals with specific maximum values. These maximum values

can be statically tailored on the command line (e.g., `-number 1000000` and `-volume 100000000`) or in the X defaults (e.g., `MTAconsole.totalnumber: 1000000` and `MTAconsole.totalvolume: 100000000`) The values can also be tailored from the badness tailoring as described in Section 7.2.4. The default values are, for volume of messages, 1000000 and, for number of messages, 10000.

Colour Configuration.

On a colour screen the MTA console will display entities of significantly different badness as different colours, with the colour ranging from green for entities with low badness to red for entities with high badness. The number of colours the console will use can be statically tailored from the command line (e.g., `-ncolours 20`) or in the Xdefaults (e.g., `MTAconsole.ncolours 20`). The console will use a range of 10 colours by default.

Static Connection Configuration.

The user must specify which QMGR to attempt to connect to. Usually this is done dynamically from the connect popup. However it can be done statically from the command line (e.g. `-Queue trellis`) or in the X defaults (e.g., `MTAconsole.connectHost: trellis`). If the QMGR to connect to is statically specified, the MTA console will automatically attempt to open an unauthorised connection to the specified QMGR.

Confirmation Configuration.

Certain of the MTA console's commands request confirmation via a popup. This feature can be turned off dynamically from the confirmation menu or statically from the command line (e.g., `-NoConfirm`) or in the X defaults (e.g., `MTAconsole.confirm: false`).

Heuristic Configuration

When in monitor mode, the MTA console displays some of the MTAs on each channel. There are different heuristics that the console can use to decide how many MTAs to display for each channel. These heuristics are:

`channelonly:` This heuristic shows only the information about the channels and no per MTA information. This is useful when the QMGR has many messages loaded into it and so refreshing the display takes a while.

`line based:` This heuristic limits the display of MTAs on each channel to a specific number of vertical lines. This number can be tailored dynamically from the configuration menu or statically on the command line (e.g., `-downLines 6`) or in the X defaults (e.g., `MTAconsole.maxvert: 6`). The default value is 4.

`percentage based:` This heuristic displays a specific percentage of the MTAs on each channel. The percentage to display is calculated by multiplying the badness of the channel (expressed as a percentage of the maximum badness) by a constant percentage. This constant percentage can be tailored dynamically from the configuration menu or statically on the command line (e.g. `-percent 70`) or in the X defaults (e.g. `MTAconsole.percentage: 70`). The default value for this constant is 50.

There is a lower bound on the number of MTAs displayed using this heuristic. The MTA console will endeavour to ensure that the number of MTAs displayed for each channel is at least the lower bound. This lower bound can be tailored dynamically from the configuration menu or statically on the command line (e.g. `-lowerbound 15`) or in the X defaults (e.g. `MTAconsole.lowerbound: 15`). The default value for this lower bound is 20.

`all:` This heuristic instructs the MTA console in monitor mode to display all the MTAs on each channel.

The choice of heuristic can be changed dynamically from the configuration menu or statically on the command line (e.g. `-heuristic all`) or in the X defaults (e.g. `MTAconsole.heuristic: all`). The line-based heuristic is used by default.

Configuration of Timeouts and Retries.

There are various timeouts and retry intervals associated with the MTA console. These can be configured or disabled from the configuration menu and are as follows:

auto-refresh timeout: If no activity occurs on the connection between the MTA console and the QMGR during a specified interval, the console will consult with the QMGR and automatically refresh the information on display. This particular timeout interval can also be configured on the command line (e.g., `-refresh 3m30s`) or in the X defaults (e.g. `MTAconsole.refresh: 3m30s`). The default interval is 5 minutes. Note: to stop the console hogging the QMGR's resources, a minimum value for this interval is enforced. This minimum is 30 seconds.

inactive timeout If the console is in *control* mode and no user input has been received during a specified interval, the MTA console automatically toggles to the *monitor* mode. The default interval is 10 minutes.

reconnection retry interval If the connection between the MTA console and the QMGR breaks, the console will attempt to reconnect to the QMGR. After an unsuccessful reconnection attempt, the console will wait for a specified time and then try to reconnect again. The intervals between reconnection attempts increases in duration by a specified increment up to a maximum interval. The initial duration of the wait